Concrete Resource Analysis of the Quantum Linear System Algorithm used to Compute the Electromagnetic Scattering Cross Section of a 2D Target

Artur Scherer, Benoît Valiron, Mau Siun-Chuon, D. Scott Alexander, Eric van den Berg, Thomas Chapuran

Quantum Inf Process (2017) 16:60
DOI 10.1007/s11128-016-1495-5

Concrete resource analysis of the quantum linear-system algorithm used to compute the electromagnetic scattering cross section of a 2D target

Artur Scherer[1,2] · Benoît Valiron[3,4] · Siun-Chuon Mau[1,5] · Scott Alexander[1] · Eric van den Berg[1] · Thomas E. Chapuran[1]

Received: 24 July 2015 / Accepted: 7 December 2016 / Published online: 25 January 2017

Abstract We provide a detailed estimate for the logical resource requirements of the quantum linear-system algorithm (Harrow et al. in Phys Rev Lett 103:150502, 2009) including the recently described elaborations and application to computing the electromagnetic scattering cross section of a metallic target (Clader et al. in Phys Rev Lett 110:250504, 2013). Our resource estimates are based on the standard quantum-circuit model of quantum computation; they comprise circuit width (related to parallelism), circuit depth (total number of steps), the number of qubits and ancilla qubits employed, and the overall number of elementary quantum gate operations as well as more specific gate counts for each elementary fault-tolerant gate from the stan-

This article is part of topical collection on *Quantum Computer Science*.

Disclaimer: The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of IARPA, DoI/NBC, or the US Government.

Electronic supplementary material The online version of this article (doi:10.1007/s11128-016-1495-5) contains supplementary material, which is available to authorized users.

✉ Artur Scherer
arturscherer17@gmail.com

1 Applied Communication Sciences, 150 Mt Airy Rd., Basking Ridge, NJ 07920, USA

2 Department of Physics and Astronomy, Macquarie University, Sydney, NSW 2109, Australia

3 CIS Department, University of Pennsylvania, 3330 Walnut Street, Philadelphia, PA 19104-6389, USA

4 Département Informatique, CentraleSupélec, Plateau de Moulon, 3 rue Joliot-Curie, 91192 Gif sur Yvette Cedex, France

5 LGS Innovations, 15 Vreeland Rd., Florham Park, NJ 07932, USA

dard set $\{X, Y, Z, H, S, T, \text{CNOT}\}$. In order to perform these estimates, we used an approach that combines manual analysis with automated estimates generated via the Quipper quantum programming language and compiler. Our estimates pertain to the explicit example problem size $N = 332{,}020{,}680$ beyond which, according to a crude big-O complexity comparison, the quantum linear-system algorithm is expected to run faster than the best known classical linear-system solving algorithm. For this problem size, a desired calculation accuracy $\varepsilon = 0.01$ requires an approximate circuit width 340 and circuit depth of order 10^{25} if oracle costs are excluded, and a circuit width and circuit depth of order 10^8 and 10^{29}, respectively, if the resource requirements of oracles are included, indicating that the commonly ignored oracle resources are considerable. In addition to providing detailed logical resource estimates, it is also the purpose of this paper to demonstrate explicitly (using a fine-grained approach rather than relying on coarse big-O asymptotic approximations) how these impressively large numbers arise with an actual circuit implementation of a quantum algorithm. While our estimates may prove to be conservative as more efficient advanced quantum-computation techniques are developed, they nevertheless provide a valid baseline for research targeting a reduction of the algorithmic-level resource requirements, implying that a reduction by many orders of magnitude is necessary for the algorithm to become practical.

Keywords Quantum computation · Concrete resource estimation in quantum algorithms · Automated quantum-circuit generation · Quantum programming language Quipper

1 Introduction

Quantum computing promises to efficiently solve certain hard computational problems for which it is believed no efficient classical algorithms exist [1]. Designing quantum algorithms with a computational complexity superior to that of their best known classical counterparts is an active research field [2]. The quantum linear-system algorithm (QLSA), first proposed by Harrow et al. [3], afterward improved by Ambainis [4], and recently generalized by Clader et al. [5], is appealing because of its great practical relevance to modern science and engineering. This quantum algorithm solves a large system of linear equations under certain conditions exponentially faster than any current classical method.

The basic idea of QLSA, essentially a matrix-inversion quantum algorithm, is to convert a system of linear equations, $A\mathbf{x} = \mathbf{b}$, where A is a Hermitian[1] $N \times N$ matrix over the field of complex numbers $\mathbb{C}$ and $\mathbf{x}, \mathbf{b} \in \mathbb{C}^N$, into an analogous quantum-theoretic version, $A\,|x\rangle = |b\rangle$, where $|x\rangle, |b\rangle$ are vectors in a Hilbert space $\mathscr{H} = (\mathbb{C}^2)^{\otimes n}$ corresponding to $n = \lceil \log_2 N \rceil$ qubits and A is a *self-adjoint* operator on $\mathscr{H}$, and use various quantum-computation techniques [1,6–8] to solve for $|x\rangle$.

Extended modifications of QLSA have also been applied to other important problems (cf. [2]), such as least-squares curve-fitting [9], solving linear differential

[1] Note that, if A is not Hermitian, the problem can be restated as $\bar{A}\bar{\mathbf{x}} = \bar{\mathbf{b}}$ with a Hermitian matrix $\bar{A} := \left(\begin{smallmatrix} 0 & A \\ A^\dagger & 0 \end{smallmatrix}\right)$, see Sect. 3.

equations [10], and machine learning [11]. Recent efforts in demonstrating small-scale experimental implementation of QLSA [12,13] have further highlighted its popularity.

1.1 Objective of this work

The main objective of this paper is to provide a *detailed logical resource estimate* (LRE) analysis of QLSA based on its further elaborated formulation [5]. Our analysis particularly also aims at including the commonly ignored resource requirements of oracle implementations. In addition to providing a detailed LRE for a large practical problem size, another important purpose of this work is to demonstrate explicitly, i.e., using a fine-grained approach rather than relying on big-O asymptotic approximations, how the concrete resource counts accumulate with an actual quantum-circuit implementation of a quantum algorithm.

Our LRE is based on an approach which combines manual analysis with automated estimates generated via the programming language *Quipper* and its compiler. Quipper [14,15] is a domain-specific, higher-order, functional language for quantum computation, embedded in the host-language Haskell. It allows automated quantum circuit generation and manipulation; equipped with a gate-count operation, Quipper offers a universal automated LRE tool. We demonstrate how Quipper's powerful capabilities have been exploited for the purpose of this work.

We underline that our research contribution is not merely providing the LRE results, but also to demonstrate an approach to how a *concrete* resource estimation can be done for a quantum algorithm used to solve a practical problem of a large size. Finally, we would also like to emphasize the *modular* nature of our approach, which allows to incorporate future work as well as to assess the impact of prospective advancements of quantum-computation techniques.

1.2 Context and setting of this work

Our analysis was performed within the scope of a larger context: *IARPA Quantum Computer Science (QCS) program* [16], whose goals were to achieve an accurate estimation and moreover a significant reduction of the necessary computational resources required to implement quantum algorithms for practically relevant problem sizes on a realistic quantum computer. The work presented here was conducted as part of our general approach to tackle the challenges of IARPA QCS program: the PLATO project,[2] which stands for "*Protocols, Languages and Tools for Resource-efficient Quantum Computation*."

The QCS program BAA [17] presented a list of seven algorithms to be analyzed. For the purpose of evaluation of the work, the algorithms were specified in "government-furnished information" (GFI) using pseudo-code to describe purely quantum subroutines and explicit oracles supplemented by Python or MATLAB code to compute parameters or oracle values. While this IARPA QCS program GFI is not

[2] The aspect of PLATO most closely aligned with the topic of this paper was the understanding of the resources required to run a quantum algorithm followed by research into the reduction of those resources.

available as published material,[3] the Quipper code developed as part of the PLATO project to implement the algorithms and used for our LRE analyses is available as published library code [18,19]. In our analyses, we found the studied algorithms to cover a wide range of different quantum-computation techniques. Additionally, with the algorithm parameters supplied for our analyses, we have seen a wide range of complexities as measured by the total number of gate operations required, including some that could not be executed within the expected life of the universe under current predictions of what a practical quantum computer would be like when it is developed.

This approach is consistent with the one commonly used in computer science for algorithms analysis. There are at least two reasons for looking at large problem sizes. First, in classical computing, we have often been wrong in trying to predict how computing resources will scale across periods of decades. We can expect to make more accurate predictions in some areas in quantum computing because we are dealing with basic physical properties that are relatively well studied. However, disruptive changes may still occur.[4] Thus, in computer science, one likes to understand the effect of scale even when it goes beyond what is currently considered practical. The second reason for considering very large problem sizes, even those beyond a practical scale, is to develop the level of abstraction necessary to cope with them. The resulting techniques are not tied to a particular size or problem and can then be adapted to a wide range of algorithms and sizes. In practice, some of our original tools and techniques were developed while expecting smaller algorithm sizes. Developing techniques for enabling us to cope with large algorithm sizes resulted in speeding up the analysis for small algorithm sizes.

Our focus in this paper is the *logical part* of the quantum algorithm implementation. More precisely, here we examine only the algorithmic-level logical resources of QLSA and do not account for all the physical overhead costs associated with techniques to enable a fault-tolerant implementation of this algorithm on a realistic quantum computer under real-world conditions. Such techniques include particularly quantum control (QC) protocols and quantum error correction (QEC) and/or mitigation codes. Nor do we take into account quantum communication costs required to establish interactions between two distant qubits so as to implement a two-qubit gate between them. These additional physical resources will depend on the actual physical realization of a quantum computer (ion traps, neutral atoms, quantum dots, superconducting qubits, photonics, etc.) and also include various other costs, such as those due to physical qubit movements in a given quantum-computer architecture, their storage in quantum memories, etc. The resource estimates provided here are for the abstract logical quantum circuit of the algorithm, assuming no errors due to real-world imperfections, no QC or QEC protocols, and no connectivity constraints for a particular physical implementation.

[3] The GFI for QLSA was provided by Clader and Jacobs, the coauthors of the work [5] whose supplementary material includes a considerable part of that GFI.

[4] At the time of ENIAC and other early classical computers, it seems unlikely that considering how the size of the computer could be reduced and its power increased would make us consider the invention of the transistor. Instead, we would have considered how vacuum tubes could be designed smaller or could be made so as to perform more complex operations.

Determining the algorithmic-level resources is a very important and indispensable first step toward a complete analysis of the overall resource requirements of each particular real-world quantum-computer implementation of an algorithm, for the following reasons. First, it helps to understand the structural features of the algorithm, and to identify the actual bottlenecks of its quantum-circuit implementation. Second, it helps to differentiate between the resource costs that are associated with the algorithmic logical-level implementation (which are estimated here) and the additional overhead costs associated with physically implementing the computation in a fault-tolerant fashion including quantum-computer-technology-specific resources. Indeed, the algorithmic-level LRE constitutes a *lower bound* on the minimum resource requirements that is independent of which QEC or QC strategies are employed to establish fault-tolerance, and independent of the physics details of the quantum-computer technology. For this reason, it is crucial to develop techniques and tools for resource-efficient quantum computation even at the logical quantum-circuit level of the algorithm implementation. The LRE for QLSA provided in this paper will serve as a *baseline* for research into the reduction of the algorithmic-level minimum resource requirements.

Finally we emphasize that our LRE analysis only addresses the resource requirements for a *single run* of QLSA, which means that it does not account for the fact that the algorithm needs to be run many times and followed by sampling in order to achieve an accurate and reliable result with high probability.

1.3 Review of previous work

The key ideas underlying QLSA [3–5] can be briefly summarized as follows; for a detailed description, see Sect. 3. The preliminary step consists of converting the given system of linear equations $A\mathbf{x} = \mathbf{b}$ (with $\mathbf{x}, \mathbf{b} \in \mathbb{C}^N$ and A a Hermitian $N \times N$ matrix with $A_{ij} \in \mathbb{C}$) into the corresponding quantum-theoretic version $A|x\rangle = |b\rangle$ over a Hilbert space $\mathscr{H} = (\mathbb{C}^2)^{\otimes n}$ of $n = \lceil \log_2 N \rceil$ qubits. It is important to formulate the original problem such that the operator $A : \mathscr{H} \to \mathscr{H}$ is *self-adjoint*, see footnote 1.

Provided that oracles exist to efficiently compute A and prepare state $|b\rangle$, the main task of QLSA is to solve for $|x\rangle$. According to the spectral theorem for self-adjoint operators, the solution can be formally expressed as $|x\rangle = A^{-1}|b\rangle = \sum_{j=1}^{N} \beta_j/\lambda_j \left|u_j\right\rangle$, where λ_j and $\left|u_j\right\rangle$ are the eigenvalues and eigenvectors of A, respectively, and $|b\rangle = \sum_{j=1}^{N} \beta_j \left|u_j\right\rangle$ is the expansion of quantum state $|b\rangle$ in terms of these eigenvectors. QLSA is designed to implement this representation.

QLSA starts with preparing (in a multiqubit data register) the known quantum state $|b\rangle$ using an oracle for vector $\mathbf{b}$. Next, Hamiltonian evolution $\exp(-iA\tau/T)$ with A as the Hamilton operator is applied to $|b\rangle$. This is accomplished by using an oracle for matrix A and Hamiltonian Simulation (HS) techniques [8]. The Hamiltonian evolution is part of the well-established technique known as *quantum phase estimation algorithm* (QPEA) [6,7], here employed as a subalgorithm of QLSA to acquire information about the eigenvalues λ_j of A and store them in QPEA's control register. In the next step, a single-qubit ancilla starting in state $|0\rangle$ is rotated by an angle inversely proportional to the eigenvalues λ_j of A stored in QPEA's control register. Finally, the latter are uncomputed by the inverse QPEA yielding a quantum state of the form

$\sum_{j=1}^{N} \beta_j \sqrt{1 - C^2/\lambda_j^2} \left|u_j\right\rangle \otimes |0\rangle + \sum_{j=1}^{N} C\beta_j/\lambda_j \left|u_j\right\rangle \otimes |1\rangle$, with the solution $|x\rangle$ correlated with the value 1 in the auxiliary single-qubit register. Thus, if the latter is measured and the value 1 is found, we know with certainty that the desired solution of the problem is stored in the quantum amplitudes of the multiqubit quantum register in which $|b\rangle$ was initially prepared. The solution can then either be revealed by an ensemble measurement (a statistical process requiring the whole procedure to be run many times), or useful information can also be obtained by computing its overlap $|\langle R|x\rangle|^2$ with a particular (known) state $|R\rangle$ (corresponding to a specific vector $\mathbf{R} \in \mathbb{C}^N$) that has been prepared in a separate quantum register [5].

Harrow, Hassidim and Lloyd (HHL) [3] showed that, given the matrix A is *well-conditioned* and *sparse* or can efficiently be decomposed into a sum of sparse matrices, and if the elements of matrix A and vector $\mathbf{b}$ can be efficiently computed, then QLSA provides an exponential speedup over the best known classical linear-system-solving algorithm. The performance of any matrix-inversion algorithm depends crucially on the *condition number* κ of the matrix A, i.e., the ratio between A's largest and smallest eigenvalues. A large condition number means that A becomes closer to a matrix which cannot be inverted, referred to as "ill-conditioned"; the lower the value of κ the more "well-conditioned" is A. Note that κ is a property of the matrix A and not of the linear-system-solving algorithm. Roughly speaking, κ characterizes the stability of the solution $\mathbf{x}$ with respect to changes in the given vector $\mathbf{b}$. Further important parameters to be taken into account are the *sparseness* d (i.e., the maximum number of nonzero entries per row/column in the matrix A), the *size* N of the square matrix A, and the desired *precision* of the calculation represented by error bound ε.

In [3] it was shown that the number of operations required for QLSA scales as

$$\widetilde{O}\left(\kappa^2 d^2 \log(N)/\varepsilon\right), \tag{1}$$

while the best known classical linear-system-solving algorithm based on conjugate gradient method [20,21] has the run-time complexity

$$O\left(Nd\kappa \log(1/\varepsilon)\right), \tag{2}$$

where, compared to $O(\cdot)$, the $\widetilde{O}(\cdot)$ notation suppresses more slowly growing terms. Thus, it was concluded in [3] that, in order to achieve an exponential speedup of QLSA over classical algorithms, κ must scale, in the worst case, as poly $\log(N)$ with the size of the $N \times N$ matrix A.

The original HHL-QLSA [3] has the drawback to be nondeterministic, because accessing information about the solution is conditional on recording outcome 1 of a measurement on an auxiliary single-qubit, thus in the worst case requiring many iterations until a successful measurement event is observed. To substantially increase the success probability for this measurement event indicating that the inversion A^{-1} has been successfully performed and the solution $|x\rangle$ (up to normalization) has been successfully computed (i.e., *probability that the postselection succeeds*), HHL-QLSA includes a procedure based on *quantum amplitude amplification* (QAA) [22]. However, in order to determine the normalization factor of the actual solution vector $|x\rangle$, the

success probability of obtaining 1 must be "measured," requiring many runs to acquire sufficient statistics. In addition, because access to the entire solution $|x\rangle$ is impractical as it is a vector in an exponentially large space, HHL suggested that the information about the solution can be extracted by calculating the expectation value $\langle x|\,\hat{M}\,|x\rangle$ of an arbitrary quantum-mechanical operator $\hat{M}$, corresponding to a quadratic form $\mathbf{x}^T M\mathbf{x}$ with some $M \in \mathbb{C}^{N\times N}$ representing the feature of $\mathbf{x}$ that one wishes to evaluate. But such a solution readout is generally also a nontrivial task and typically would require the whole algorithm to be repeated numerous times.

In a subsequent work, Ambainis [4] proposed using *variable-time quantum amplitude amplification* to improve the run-time of HHL algorithm from $\widetilde{O}(\kappa^2 \log N)$ to $\widetilde{O}(\kappa \log^3 \kappa \log N)$, thus achieving an almost optimal dependence on the condition number κ.[5] However, the improvement of the dependence of the run-time on κ was thereby attained at the cost of substantially worsening its scaling in the error bound ε.

The recent QLSA analysis by Clader, Jacobs and Sprouse (CJS) [5] incorporates useful generalizations to make the original algorithm more practical. In particular, a general method is provided for efficient preparation of the generic quantum state $|b\rangle$ (as well as of $|R\rangle$). Moreover, CJS proposed a deterministic version of the algorithm by removing the postselection step and demonstrating a resolution to the read-out problem discussed above. This was achieved by introducing several additional single-qubit ancillae and using the *quantum amplitude estimation* (QAE) technique [22] to deterministically estimate the values of the success probabilities of certain ancillae measurement events in terms of which the overlap $|\langle R\,|x\rangle|^2$ of the solution $|x\rangle$ with any generic state $|R\rangle$ can be expressed after performing a controlled swap operation between the registers storing these vectors. Finally, CJS also addressed the condition-number scaling problem and showed how by incorporating *matrix preconditioning* into QLSA, the class of problems that can be solved with exponential speedup can be expanded to worse than $\kappa \sim \text{poly}\log(N)$-conditioned matrices. With these generalizations aiming at improving the efficiency and practicality of the algorithm, CJS-QLSA was shown to have the run-time complexity[6]

$$\widetilde{O}\left(\kappa d^7 \log(N)/\varepsilon^2\right), \tag{3}$$

[5] In [3] it was also shown that the run-time cannot be made poly $\log(\kappa)$, unless **BQP** = **PSPACE**, which, while not yet disproven, is highly unlikely to be true in computational complexity theory. Hence, because poly $\log(\kappa) = o(\kappa^{\varepsilon})$ for all $\varepsilon > 0$, QLSA's run-time is asymptotically also bounded from below as given by complexity $\Omega(\kappa^{1-o(1)})$.

[6] But note that, while the CJS run-time complexity [Eq. (3)] scales quadratically better in the condition number κ than the original HHL complexity [Eq. (1)], the former scales quadratically worse than the latter with respect to the parameters d and ε. However, the two run-time complexities should not be directly compared, because the corresponding QLS algorithms achieve somewhat different tasks. Besides, it is our opinion that the linear scaling of CJS run-time complexity in κ is based on an overoptimistic assumption in its derivation. Indeed, while CJS removed the QAA step from the HHL algorithm, they replaced it with the nearly equivalent QAE step, which we believe has a similar resource requirement as the former, and thus may require up to $O(\kappa/\varepsilon)$ iterations to ensure successful amplitude estimation within multiplicative accuracy ε, in addition to the factor $O(\kappa/\varepsilon)$ resulting from the totally independent QPEA step. See also our remark in footnote 11.

which is quadratically better in κ than in the original HHL-QLSA. To demonstrate their method, CJS applied QLSA to computing the electromagnetic scattering cross section of an arbitrary object, using the finite-element method (FEM) to transform Maxwell's equations into a sparse linear system [23,24].

1.4 What makes our approach differ from previous work?

In the previous analyses of QLSA [3–5], resource estimation was performed using "big-O" complexity analysis, which means that it only addressed the *asymptotic behavior* of the *run-time* of QLSA, with reference to a similar big-O characterization for the best known classical linear-system-solving algorithm. Big-O complexity analysis is a fundamental technique that is widely used in computer science to classify algorithms; indeed, it represents the core characterization of the most significant features of an algorithm, both in classical and quantum computing. This technique is critical to understanding how the use of resources and time grows as the inputs to an algorithm grow. It is particularly useful for comparing algorithms in a way where details, such as start-up costs, do not eclipse the costs that become important for the larger problems where resource usage typically matters. However, this analysis assumes that those constant costs are dwarfed by the asymptotic costs for problems of interest as has typically proven true for practical classical algorithms. In QCS, we set out to additionally learn (1) whether this assumption holds true for quantum algorithms, and (2) what the actual resource requirements would be as part of starting to understand what would be required for a quantum computer to be a *practical* quantum computer.

In spite of its key relevance for analyzing algorithmic efficiency, a big-O analysis is not designed to provide a detailed accounting of the resources required for any specific problem size. That is not its purpose, rather it is focused on determining the asymptotic leading-order behavior of a function, and does not account for the constant factors multiplying the various terms in the function. In contrast, in our case we are interested, for a specific problem input size, in detailed information on such aspects as the number of qubits required, the size of the quantum circuit, and run-time required for the algorithm. These aspects, in turn, are critical to evaluating the practicality of actually implementing the algorithm on a quantum computer.

Thus, in this work we report a detailed analysis of the number of qubits required, the quantity of each type of elementary quantum logic gate, the width and depth of the quantum circuit, and the number of logical timesteps needed to run the algorithm—all for a realistic set of parameters κ, d, N, and ε. Such a fine-grained approach to a *concrete* resource estimation may help to identify the actual bottlenecks in the computation, which algorithm optimizations should particularly focus on. Note that this is similar to the practice in classical computing, where we would typically use techniques like run-time profiling to determine algorithmic bottlenecks for the purpose of program optimization. It goes without much saying that the big-O analyses in [3–5] and the more fine-grained LRE analysis approach presented here are both valuable and complement each other.

Two more differences are worth mentioning. Unlike in previous analyses of QLSA, our LRE analysis particularly also includes resource requirements of ora-

cle implementations. Finally, this work leverages the use of novel universal automated circuit-generation and resource-counting tools (e.g., Quipper) that are currently being developed for resource-efficient implementations of quantum computation. As such our work advances efforts and techniques toward practical implementations of QLSA and other quantum algorithms.

1.5 Main results of this work

We find that surprisingly large logical gate counts and circuit depth would be required for QLSA to exceed the performance of a classical linear-system-solving algorithm. Our estimates pertain to the specific problem size $N = 332{,}020{,}680$. This explicit example problem size has been chosen such that QLSA and the best known classical linear-system-solving method are expected to require roughly the same number of operations to solve the problem, assuming equal algorithmic precisions. This is obtained by comparing the corresponding big-O estimates, Eqs. (3) and (2). Thus, beyond this "cross-over point" the quantum algorithm is expected to run faster than any classical linear-system-solving algorithm. Assuming an algorithmic accuracy $\varepsilon = 0.01$, gate counts and circuit depth of order 10^{29} or 10^{25} are found, respectively, depending on whether we take the resource requirements for oracle implementations into account or not, while the numbers of qubits used simultaneously amount to 10^8 or 340, respectively. These numbers are several orders of magnitude larger than we had initially expected according to the big-O analyses in [3,5], indicating that the constant factors (which are not included in the asymptotic big-O estimates) must be large. This indicates that more research is needed about whether asymptotic analysis needs to be supplemented, particularly in comparing quantum to classical algorithms.

To get an idea of our results' implications, we note that the practicality of implementing a quantum algorithm can strongly be affected by the number of qubits and quantum gates required. For example, the algorithm's run-time crucially depends on the circuit depth. With circuit depth on the order of 10^{25}, and with gate operation times of 1 ns (as an example), the computation would take approx. 3×10^8 years. And such large resource estimates arise for the solely logical part of the algorithm implementation, i.e., even assuming perfect gate performance and ignoring the additional physical overhead costs (associated with QEC/QC to achieve fault-tolerance and specifics of quantum-computer technology). In practice, the full physical resource estimates typically will be even larger by several orders of magnitude.

One of the main purposes of this paper is to demonstrate how the impressively large LRE numbers arise and to explain the actual bottlenecks in the computation. We find that the dominant resource-consuming part of QLSA is Hamiltonian Simulation and the accompanying quantum-circuit implementations of the oracle queries associated with Hamiltonian matrix A. Indeed, to be able to accurately implement each run of the Hamiltonian evolution as part of QPEA, one requires a large time-splitting factor of order 10^{12} when utilizing the Suzuki-Higher-Order Integrator method including Trotterization [8,25,26]. And each single timestep involves numerous oracle queries for matrix A, where each query's quantum-circuit implementation yields a further factor of several orders of magnitude for gate count. Hence, our LRE results suggest that

the resource requirements of QLSA are to a large extent dominated by the numerous oracle A queries and their associated resource demands. Finally, our results also reveal lack of parallelism; the algorithmic structure of QLSA is such that most gates must be performed successively rather than in parallel.

Our LRE results are intended to serve as a *baseline* for research into the reduction of the logical resource requirements of QLSA. Indeed, we anticipate that our estimates may prove to be conservative as more efficient quantum-computation techniques become available. However, these estimates indicate that, for QLSA to become practical (i.e., its implementation in real world to be viable for relevant problem sizes), a resource reduction by many orders of magnitude is necessary (as is, e.g., suggested by $\sim 3 \times 10^8$ years for the optimistic estimate of the run-time given current knowledge).

1.6 Outline of the paper

This paper is organized as follows. In Sect. 2 we identify the resources to be estimated and expand on our goals and techniques used. In Sect. 3 we describe the structure of QLSA and elaborate on its coarse-grained profiling with respect to resources it consumes. Section 4 demonstrates our quantum implementation of oracles and the corresponding automated resource estimation using our quantum programming language Quipper (and compiler). Our LRE results are presented in Sect. 5 and further reviewed in Sect. 6. We conclude with a brief summary and discussion in Sect. 7.

2 Resource estimation

As mentioned previously, the main goal of this work is to find concrete logical resource estimates of QLSA as accurately as possible, for a problem size for which the quantum algorithm and the best known classical linear-system-solving algorithm are expected to require a similar run-time order of magnitude, and beyond which the former provides an exponential speedup over the latter. An approximation for this specific "cross-over point" problem size can be derived by comparing the coarse run-time big-O estimates of the classical and quantum algorithms, provided, respectively, by Eqs. (2) and (3), assuming the same algorithmic computation precision ε, and the same κ and d values.[7] For instance, choosing the accuracy $\varepsilon = 0.01$ and presuming $d \approx 10$, yields the approximate value $N_{\text{cross}} \approx 4 \times 10^7$ for the cross-over point. The specified example problem that has been subject to our LRE analysis has the somewhat larger size $N = 332{,}020{,}680$, while the other relevant parameters have the values $\kappa = 10^4$, $d = 7$, and $\varepsilon = 10^{-2}$.

Logical resources to be tracked are the overall *number of qubits* (whereby we track data qubits and ancilla qubits separately), *circuit width* (i.e., the max. number of qubits in use at a time, which also corresponds to the max. number of "wires" in algorithm's circuit), *circuit depth* (i.e., the total number of logical steps specifying the length of the longest path through the algorithm's circuit assuming maximum parallelism), the

[7] Note that the run-time big-O estimates of the classical [Eq. (2)] and the quantum [Eq. (3)] algorithm both scale linearly with κ.

number of *elementary (1- and 2-qubit) gate operations* (thereby tracking the quantity of each particular type of gate operation), and "*T-depth*" (i.e., the total number of logical steps containing at least one T-gate operation, meaning the total number of T-gate operations that cannot be performed in parallel but must be implemented successively in series). While we are not considering the costs of QEC in this paper, it is nevertheless important to know that, when QEC is considered, the T gate, as a *nontransversal gate*, has a much higher per-gate resource cost than the transversal gates X, Y, Z, H, S, and CNOT, and thus contributes more to algorithm resources relative to the latter. It is for this reason that we call out the T-depth separately.

Note that the analysis in this paper involves only the abstract algorithmic-level logical resources; i.e., we ignore all additional costs that must be taken into account when implementing the algorithm on a fault-tolerant real-world quantum computer, namely resources associated with techniques to avoid, mitigate, or correct errors which occur due to decoherence and noise. More specifically, here we omit the overhead resource costs associated with various QC and QEC strategies. We furthermore assume no connectivity constraints, thus ignoring resources needed to establish fault-tolerant quantum communication channels between two distant (physically remotely located) qubits which need to interact in order to implement a two-qubit gate such as a CNOT in the course of the algorithm implementation. Besides being an indispensable first step toward a complete resource analysis of any quantum algorithm, focusing on the algorithmic-level resources allows setting a lower limit on resource demands which is independent of the details of QEC approaches and physical implementations, such as qubit technology.

To be able to represent large circuits and determine estimates of their resource requirements, we take advantage of repetitive patterns and the hierarchical nature of circuit decomposition down to elementary quantum gates and its associated *coarse-grained profiling* of logical resources. For example, we generate "*templates*" representing circuit blocks that are reused frequently, again and again. These templates capture both the quantum circuits of the corresponding algorithmic building blocks (subroutines or multiqubit gates) and their associated resource counts. As an example, it is useful to have a template for Quantum Fourier Transform (or its inverse) acting on n qubits; for other templates, see Fig. 2 and "Appendix 2." The cost of a subroutine may thereby be measured in terms of the number of specified gates, data qubits, ancilla uses, etc., or/and in addition in terms of calls of lower-level subsubroutines and their associated costs. Furthermore, the cost may vary depending on input argument value to the subroutine. Many of the intermediate steps represent multiqubit gates that are frequently used within the overall circuit. Such *intermediate representations* can therefore also improve the efficiency of data representation. Accordingly, each higher-level circuit block is decomposed in a hierarchical fashion, in a series of steps, down to elementary gates from the standard set $\{X, Y, Z, H, S, T, \text{CNOT}\}$, using the decomposition rules for circuit templates (see "Appendices 1 and 2" for details).

Indeed, QLSA works with many repetitive patterns of quantum circuits involving numerous iterative operations, repeated a large number of times. Repetitive patterns arise from the well-established techniques such as Quantum Phase Estimation, Quantum Amplitude Estimation, and Hamiltonian Simulation based on Suzuki-

Higher-Order Integrator decomposition and Trotterization. These techniques involve large iterative factors, thus contributing many orders of magnitude to resource requirements, in particular to the circuit depth. Indeed, these large iterative factors explain why we get such large gate counts and circuit depth.

It is useful to differentiate between the resources associated with the "*bare algorithm*" excluding oracle implementations and those which also include the implementation of oracles. In order to perform the LRE, we chose an approach which combines manual analysis for the bare algorithm ignoring the cost of oracle implementations (see Sect. 3) with automated resource estimates for oracles generated via the Quipper programming language and compiler (see Sect. 4). Whereas a manual LRE analysis was feasible for the bare algorithm thus allowing a better understanding of its structural "profiling" as well as checking the reliability of the automated resource counts, it was not feasible (or too cumbersome) for the oracle implementations. Hence, an automated LRE was inevitable for the latter. The Quipper programming language is thereby demonstrated as a universal automated resource estimation tool.

3 Quantum linear-system algorithm and its profiling

3.1 General remarks

QLSA computes the solution of a system of linear equations, $A\mathbf{x} = \mathbf{b}$, where A is a Hermitian $N \times N$ matrix over $\mathbb{C}$ and $\mathbf{x}, \mathbf{b} \in \mathbb{C}^N$. For this purpose, the (classical) linear system is converted into the corresponding quantum-theoretic analogue, $A\,|x\rangle = |b\rangle$, where $|x\rangle\,, |b\rangle$ are vectors in a Hilbert space $\mathscr{H} = (\mathbb{C}^2)^{\otimes n}$ corresponding to $n = \lceil \log_2 N \rceil$ qubits and A is a Hermitian operator on $\mathscr{H}$. Note that, if A is not Hermitian, we can define $\bar{A} := \left(\begin{smallmatrix} 0 & A \\ A^\dagger & 0 \end{smallmatrix}\right)$, $\bar{\mathbf{b}} := (\mathbf{b}, 0)^T$, and $\bar{\mathbf{x}} := (0, \mathbf{x})^T$, and restate the problem as $\bar{A}\bar{\mathbf{x}} = \bar{\mathbf{b}}$ with a Hermitian $2N \times 2N$ matrix $\bar{A}$ and $\bar{\mathbf{x}}, \bar{\mathbf{b}} \in \mathbb{C}^{2N}$.

The basic idea of QLSA has been outlined in the Introduction. In what follows, we illustrate the structure of QLSA including the recently proposed generalization [5] in more detail. In particular, we expand on its *coarse-grained profiling* with respect to resources it consumes. Our focus in this section is the implementation of the bare algorithm, which accounts for oracles only in terms of the number of times they are queried. The actual quantum-circuit implementation of oracles is presented in Sect. 4. Our overall LRE results are summarized in Sect. 5.

3.2 Problem specification

We analyze a concrete example which was demonstrated as an important QLSA application of high practical relevance in [5]: the linear system $A\mathbf{x} = \mathbf{b}$ arising from solving Maxwell's equations to determine the electromagnetic scattering cross section of a specified target object via the *Finite-Element Method* (FEM) [23]. Applied in sciences and engineering as a numerical technique for finding approximate solutions to boundary-value problems for differential equations, FEM often yields linear systems $A\mathbf{x} = \mathbf{b}$ with highly *sparse* matrices—a necessary condition for QLSA. The FEM

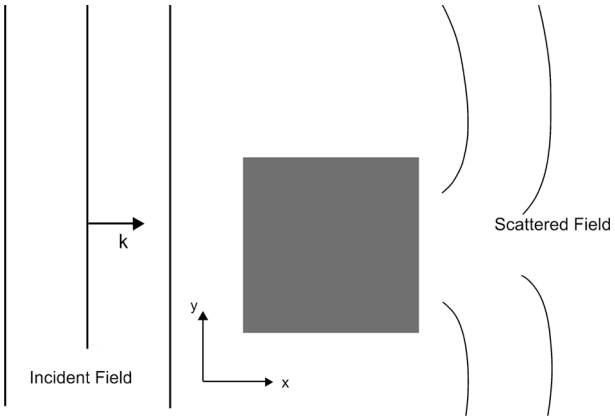

Fig. 1 A 2D toy-problem: scattering of a linearly polarized plane electromagnetic wave off a metallic object with a 2-dimensional scattering geometry. A simple square was chosen for our example problem, with edges of length $L = 2\lambda$ aligned with the Cartesian x–y plane axes, and an incident field with wavelength λ and wave vector $\boldsymbol{k} = (2\pi/\lambda)\boldsymbol{e}_x$ propagating toward the square. When interacting with the metallic object the electromagnetic wave scatters off into all directions. The task consists in computing the far-field radar cross section using the FEM approach to solve Maxwell's equations

approach to solving Maxwell's equations for scattering of electromagnetic waves off an object, as demonstrated in [5,23,24], introduces a discretization by breaking up the computational domain into small volume elements and applying boundary conditions at neighboring elements. Using finite-element edge basis vectors [24], the system of differential Maxwell's equations is thereby transformed into a sparse linear system. The matrix A and vector $\mathbf{b}$ comprise information about the scattering object; they can be derived, and efficiently computed, from a functional that depends only on the discretization chosen and the boundary conditions which account for the scattering geometry. For details, see [5] and [23,24] including its supplementary material.

Within the scope of the PLATO project, we analyzed a 2D toy-problem given by scattering of a linearly polarized plane electromagnetic wave $\boldsymbol{E}(x, y) = E_0 \boldsymbol{p} \exp[i(\boldsymbol{k} \cdot \boldsymbol{r} - \omega t)]$, with magnitude E_0, frequency ω, wave vector $\boldsymbol{k} = k(\cos\theta \boldsymbol{e}_x + \sin\theta \boldsymbol{e}_y)$, and polarization unit vector $\boldsymbol{p} = \boldsymbol{e}_z \times \boldsymbol{k}/k$, while $\boldsymbol{r} = x\boldsymbol{e}_x + y\boldsymbol{e}_y$ is the position, off a metallic object with a 2-dimensional scattering geometry. The scattering region can have any arbitrary design. A simple square shape was specified for our example problem, whose edges are parallel (or perpendicular) to the Cartesian x-y plane axes, and an incident field propagating in x-direction ($\theta = 0$) toward the square, as illustrated in Fig. 1. The receiver polarization, needed to calculate the far-field radar cross section of the scattered waves, has been assumed to be parallel to the polarization of the incident field.

For the sake of simplicity, for FEM analysis we used a two-dimensional uniform finite-element mesh with *square* finite elements. Note that QLSA requires the matrix elements to be efficiently computable, a constraint which restricts the class of FEM meshes that can be employed. As a result of the local nature of the finite-element expansion of the scattering problem, the corresponding linear system has a highly sparse matrix A. For meshes with rectangular finite elements, the maximum number of nonzero elements in each row of A (i.e., sparseness) is $d = 7$. Moreover, for regular

grids, such as used for our analysis, we obtain a *banded* sparse matrix A, with a total of $N_b = 9$ bands.

The actual instructions for computing the elements of the linear system's matrix A and vector $\mathbf{b}$, as well as of the vector whose overlap with the solution $\mathbf{x}$ is used to calculate the far-field radar cross section (see Sect. 3.3), are specified in our Quipper code for QLSA, see [18,19]. The metallic scattering region is thereby given in terms of an array of scatteringnodes denoted as "`scatteringnodes`." Here we briefly summarize the FEM dimensions and the values of all other system parameters that are necessary to reproduce the analysis. For all other details, we refer the reader to our QLSA's Quipper code and its documentation in [18,19].

The total number of FEM vertices in x and y dimensions were $n_x = 12{,}885$ and $n_y = 12{,}885$, respectively, yielding $N = n_x(n_y - 1) + n_y(n_x - 1) = 332{,}020{,}680$ for the total number of FEM edges, which thus determines the number of edge basis vectors, and hence also the size of the linear system, and in particular the size of the $N \times N$ matrix A. The lengths of FEM edges in x and y dimensions were $l_x = 0.1m$ and $l_y = 0.1m$, respectively. The analyzed 2D scattering object was a square with edge length $L = 2\lambda$, which in our analysis was placed right in the center of the FEM grid. In our Quipper code for QLSA [18,19] it is represented by the array "`scatteringnodes`" containing the corner vertices of the scattering region. The dimensions of the scattering region can also be expressed in terms of the number of vertices in x and y directions; using $\lambda = 1m$ (see below), the scatterer was given by a 200×200 square area of vertices. The incident and scattered field parameters were specified as follows. The incident field amplitude, wave number and angle of incidence were set $E_0 = 1.0\,V/m, k = 2\pi\, m^{-1}$ (implying wavelength $\lambda = 1m$) and $\theta = 0$, respectively. The receiver (for scattered field detection) was assumed to have the same polarization direction as the incident field and located along the x-axis (at angle $\phi = 0$). The task of QLSA is to compute the far-field radar cross section with a precision specified in terms of the multiplicative error bound $\varepsilon = 0.01$.

Finally, we remark that our example analysis does not include matrix preconditioning that was also proposed in [5] to expand the number of problems that can achieve exponential speedup over classical linear-system algorithms. With no preconditioning, condition numbers of the linear-system matrices representing a finite-element discretization of a boundary-value problem typically scale worse than poly-log(N), which would be necessary to attain a quantum advantage over classical algorithms. Indeed, as was rigorously proven in [27,28], FEM matrix condition numbers are generally bounded from above by $O(N^{2/n})$ for $n \geq 3$ and by $\widetilde{O}(N)$ for $n = 2$, with n the number of dimensions of the problem. For regular meshes, the bound $O(N^{2/n})$ is valid for all $n \geq 2$. In our 2D toy-problem, $n = 2$ and the mesh is regular, implying that the condition number is bounded by $O(N)$. However, we used the much smaller value $\kappa = 10^4$ from IARPA GFI to perform our LRE. This "guess" can be motivated by an estimate for the lower bound of κ that we obtained numerically.[8]

[8] The condition number of a matrix A is defined by $\kappa_p(A) = \|A\|_p \|A^{-1}\|_p$, where $\|\cdot\|_p$ denotes the matrix norm that is used to induce a metric. Hence, the condition number is also a function of the norm which is used. The 1-norm $\|\cdot\|_1$ and 2-norm $\|\cdot\|_2$ are commonly used to define the condition number,

3.3 QLSA: abstract description

The generalized QLSA [5] is based on two well-known quantum algorithm techniques: (1) *Quantum Phase Estimation Algorithm* (QPEA) [6,7], which uses *Quantum Fourier Transform* (QFT) [1] as well as *Hamiltonian Simulation* (HS) [8] as quantum computational primitives, and (2) *Quantum Amplitude Estimation Algorithm* (QAEA) [22], which uses *Grover's search-algorithm* primitive. The purpose of QPEA, as part of QLSA, is to gain information about the eigenvalues of the matrix A and move them into a quantum register. The purpose of the QAEA procedure is to avoid the use of nondeterministic (nonunitary) measurement and postselection processes by estimating the quantum amplitudes of the desired parts of quantum states, which occur as superpositions of a "good" part and a "bad" part.[9]

QLSA requires several quantum registers of various sizes, which depend on the problem size N and/or the precision ε to which the solution is to be computed. We denote the jth quantum register by R_j, its size by n_j, and the quantum state corresponding to register R_j by $|\psi\rangle_j$ (where ψ is a label for the state). The following Table 1 lists all logical qubit registers that are employed by QLSA, specified by their size as well as purpose. The register size values chosen (provided in GFI within the scope of IARPA QCS program) correspond to the problem size $N = 332{,}020{,}680$ and algorithm precision $\varepsilon = 0.01$.

For example, the choice $n_0 = \lceil \log_2 M \rceil = 14$ for the size of the QAE control register can be explained as follows. According to the error analysis of Theorem 12 in [22], using QAEA the modulus squared $0 \leq \alpha \leq 1$ of a quantum amplitude can be estimated within $\pm\varepsilon\alpha$ of its correct value[10] with a probability at least $8/\pi^2$ for $k = 1$ and with a probability greater than $1 - \frac{1}{2(k-1)}$ for $k \geq 2$, if the QAE control register's

Footnote 8 continued
and obviously $\kappa_1 \neq \kappa_2$ in general. But due to $\|A\|_1/\sqrt{N} \leq \|A\|_2 \leq \sqrt{N}\|A\|_1$ for $N \times N$ matrices A, knowing the condition number for either of these two norms allows to bound the other. Furthermore, if A is normal (i.e., diagonalizable and has a spectral decomposition), then $\kappa_2 = |\lambda_{\max}|/|\lambda_{\min}|$, where $\lambda_{\max}$ and $\lambda_{\min}$ are the maximum and minimum eigenvalues of A. For a regular mesh of *size* h, κ_2 generally scales as $O(h^{-2})$ [27–29]. Hence, because the number of degrees of freedom scales as $N = O(h^{-n})$, κ_2 is bounded by $O(N^{2/n})$ (see [27,28] for rigorous proof). In our toy-problem, $h \approx 0.1$ whereas $N \approx 3 \times 10^8$, thus it is not evident whether a guess for κ_2 should be based on $O(h^{-2})$ or $O(N)$, as the two bounds indeed differ by many orders of magnitude. Besides, as our LRE analysis aims at achieving an optimistic (as opposed to an overly conservative) resource count for QLSA, it is more sensible to use the lower bound rather than the upper bound as a guess for κ_2. Hence, we attempted to find an actual lower bound for κ_2 numerically. To this end, because an estimate for κ_1 can be obtained with much less computational expense than for κ_2 for a given matrix of a very large size, we used MATLAB and extrapolation techniques to attain a rough approximation of κ_1 from the given code specifying the matrix of our toy-problem. We found a value $\kappa_1 \approx 10^7$. This allowed us to infer a rough estimate for the lower bound for κ_2. Indeed, using the above relation between the matrix norms $\|\cdot\|_1$ and $\|\cdot\|_2$ for a square matrix and realizing that both $\|A\|_1$ and $\|A\|_2$ have values of order $O(1)$, we may conclude that $\kappa_2 \geq \kappa_1/\sqrt{N} \times O(1)$, which is of order approximately $10^3 - 10^4$.

[9] Let $|\psi\rangle = |\psi_{\text{good}}\rangle + |\psi_{\text{junk}}\rangle$ be a superposition of the good and the junk components of a (normalized) quantum state $|\psi\rangle$. The goal of QAEA [22] is to estimate $\alpha := \langle\psi_{\text{good}}|\,|\psi_{\text{good}}\rangle$, i.e., the modulus squared of the amplitude of the desired good component.

[10] Note that we hereby use a *multiplicative error* bound to represent the desired precision of QAEA's computation.

Hilbert space dimension M is chosen such that (see [22])

$$|\tilde{\alpha} - \alpha| \leq \frac{2k\pi\sqrt{\alpha(1-\alpha)}}{M} + \frac{k^2\pi^2}{M^2} \leq \varepsilon\alpha, \tag{4}$$

where $\tilde{\alpha}$ ($0 \leq \tilde{\alpha} \leq 1$) denotes the output of QAEA. Moreover, if $\alpha = 0$ then $\tilde{\alpha} = 0$ with certainty, and if $\alpha = 1$ and M is even, then $\tilde{\alpha} = 1$ with certainty. Corollary (4) can be viewed as a requirement used to determine the necessary value of M, yielding (for $\alpha \neq 0$)

$$M \geq \left\lceil \frac{k\pi}{\varepsilon\sqrt{\alpha}} \left(\sqrt{1-\alpha} + \sqrt{1-\alpha+\varepsilon} \right) \right\rceil. \tag{5}$$

The RHS of this expression is strictly decreasing, tending to $\frac{k\pi}{\sqrt{\varepsilon\alpha}}$ as α becomes close to 1, whereas for $\alpha \ll 1$ we have $M \geq \lceil \frac{k\pi}{\varepsilon\sqrt{\alpha}}[(1 - \frac{\alpha}{2}) + (1 - \frac{\alpha-\varepsilon}{2})] \rceil = \lceil \frac{2k\pi}{\varepsilon\sqrt{\alpha}} \rceil$. Hence, we take $M \geq \lceil \frac{2k\pi}{\varepsilon\sqrt{\alpha}} \rceil$, so as to account for all possibilities. Moreover, we want QAEA to succeed with a probability close to 1, allowing failure only with a small error probability $\wp_{\text{err}}$. According to Theorem 12 in [22], this indeed can be achieved when $1 - \frac{1}{2(k-1)} \geq 1 - \wp_{\text{err}}$, i.e., for $k \geq \lceil 1 + \frac{1}{2\wp_{\text{err}}} \rceil$, and thus for

$$M \geq \left\lceil \frac{\pi}{\varepsilon\sqrt{\alpha}} \left(2 + \frac{1}{\wp_{\text{err}}} \right) \right\rceil. \tag{6}$$

While we may assume any value for the failure probability, for the sake of simplicity we here choose $\wp_{\text{err}} = \varepsilon$, which is also the desired precision of QLSA. Unless α is very small, this justifies our choice $M = 2^{\lceil \log_2(1/\varepsilon^2) \rceil}$. A similar requirement for the value of M was also proposed in the supplementary material of [5]. In our example computation, $\varepsilon = 0.01$, and so we have $n_0 = 14$. Note that small α values require an even larger value for the QAE control register size in order to ensure that the estimate $\tilde{\alpha}$ is within $\pm\varepsilon\alpha$ of the actual correct value with a success probability greater than $1 - \varepsilon$.

As a *first step*, QLSA *prepares* the known quantum state $|b\rangle_2 = \sum_{j=0}^{N-1} b_j |j\rangle_2$ in a multiqubit quantum *data register* R_2 consisting of $n_2 = \lceil \log_2(2N) \rceil$ qubits. This step requires numerous queries (see details below) of an oracle for vector **b**. Moreover, as pointed out in [5], efficient quantum state preparation of arbitrary states is in general not always possible. However, the procedure proposed in [5] can efficiently generate the state

$$|b_T\rangle_{2,6} = \cos(\phi_b) \left|\tilde{b}\right\rangle_2 \otimes |0\rangle_6 + \sin(\phi_b) |b\rangle_2 \otimes |1\rangle_6 , \tag{7}$$

where the multiqubit *data register* R_2 contains (as a quantum superposition) the desired arbitrary state $|b\rangle$ entangled with a 1 in an auxiliary single-qubit register R_6, as well as a garbage state $\left|\tilde{b}\right\rangle$ (denoted by the tilde) entangled with a 0 in register R_6. To generate the state (7), in addition to data registers R_2 and single-qubit auxiliary register R_6, two further, computational registers R_4 and R_5 are employed, each consisting of n_4 auxiliary qubits. The latter registers are used to store the magnitude and phase components, which in [5] are denoted as b_j and ϕ_j, respectively, that are computed each time the oracle b is queried. Which component ($j = 1, 2, 3, \ldots$) to query is

Table 1 QLSA logical qubit registers specified by their size and purpose

Qubit register	Size	Purpose
R_0	$n_0 = \lceil \log_2 M \rceil = 14$	QAE control register
R_1	$n_1 = \lceil \log_2 T \rceil = 24$	HS control register
R_2, R_3	$n_2 = n_3 = \lceil \log_2(2N) \rceil = 30$	Quantum data register
R_4, R_5	$n_4 = n_5 = 65$	Computational register
$R_6, \ldots, R_{10}$	1	Single-qubit ancilla
R_{11}	$n_1 = \lceil \log_2 T \rceil = 24$	Auxiliary register for IntegerInverse
R_{12}	$n_2 = \lceil \log_2(2N) \rceil = 30$	Auxiliary register for HS subroutines

The parameters M and T characterize the precision of the QAE and QPE procedure, respectively. According to the error analysis in [22], choosing $M = 2^{\lceil \log_2(1/\varepsilon^2) \rceil}$ ensures that the modulus squared α of a quantum amplitude can be estimated by QAEA with a probability greater than $1 - \varepsilon$ within $\pm\varepsilon\alpha$ of its correct value, with ε specifying the desired precision, which in our analysis is chosen to be 0.01. Registers R_2 and R_3 are used for storing and processing the quantum data such as $|b\rangle$, $|x\rangle$, and $|R\rangle$. Computational registers R_4 and R_5 are used to hold signed integer values, where the last bit is the sign bit, with the convention that 0 stands for a positive number and 1 for a negative number, respectively. Several single-qubit auxiliary (ancilla) registers $R_6 \ldots, R_{10}$ are employed throughout the algorithm. In addition, an n_1-qubit ancilla register R_{11} is needed to store the *inverse* values λ_j^{-1} of the eigenvalues of matrix A, and a further n_2-qubit ancilla register R_{12} must be employed as part of HS subroutines

thereby controlled by data register R_2. The quantum circuit for state preparation [Eq. (7)] is shown in Sect. 3.4.3, Fig. 13. Following the oracle b queries, a controlled-phase gate is applied to the auxiliary single-qubit register R_6, controlled by the calculated value of the phase carried by quantum register R_5; in addition, the single-qubit register R_6 is rotated conditioned on the calculated value of the amplitude carried by quantum register R_4. Uncomputing registers R_4 and R_5 involves further oracle b calls, leaving registers R_2 and R_6 in the state (7) with $\sin^2\phi_b = \frac{C_b^2}{2N}\sum_{j=0}^{2N-1} b_j^2$ and $\cos^2\phi_b = \frac{1}{2N}\sum_{j=0}^{2N-1}\left(1 - C_b^2 b_j^2\right)$, where $C_b = 1/\max(b_j)$, cf. [5].

As a *second step*, QPEA is employed to acquire information about the eigenvalues λ_j of A and store them in a multiqubit *control register* R_1 consisting of $n_1 = \lceil \log_2 T \rceil$ qubits, where the parameter T characterizes the precision of the QPEA subroutine and is specified in Table 1. This high-level step consists of several hierarchy levels of lower-level subroutines decomposing it down to a fine-grained structure involving only elementary gates. More specifically, controlled Hamiltonian evolution $\sum_{\tau=0}^{T-1} (|\tau\rangle\langle\tau|)_1 \otimes \left[\exp(-iA\tau t_0/T)\right]_2 \otimes \mathbb{1}_6$ with A as the Hamiltonian is applied to quantum state $|\phi\rangle_1 \otimes |b_T\rangle_{2,6}$. Here, similar to the presentation in [3], a time constant t_0 such that $t = \tau t_0/T \leq t_0$ has been introduced for the purpose of minimizing the error for a given condition number κ and matrix norm $\|A\|$. As shown in [3], for the QPEA to be accurate up to error $O(\varepsilon)$, we must have $t_0 \sim O(\kappa/\varepsilon)$ if $\|A\| \sim O(1)$. Accordingly, we define $t_0 := \|A\|\kappa/\varepsilon$. The application of $\exp(-iA\tau t_0/T)$ on the data register R_2 is thereby controlled by n_1-qubit control register R_1 prepared in state $|\phi\rangle_1 = H^{\otimes n_1}|0\rangle^{\otimes n_1} = \frac{1}{\sqrt{T}}\sum_{\tau=0}^{T-1}|\tau\rangle_1$ (with H denoting the Hadamard gate). Controlled Hamiltonian evolution is subsequently followed by a QFT of register R_1 to complete QPEA.

The Hamiltonian quantum state evolution is accomplished by multiquerying an oracle for matrix A and HS techniques [8], which particularly include the decomposition

of the Hamiltonian matrix into a sum

$$A = \sum_{j=1}^{m} A_j \tag{8}$$

of submatrices, each of which ought to be *1-sparse*, as well as the *Suzuki-Higher-Order Integrator* method and *Trotterization* [25,26]. In the general case, an arbitrary sparse matrix A with sparseness d can be decomposed into $m = 6d^2$ 1-sparse matrices A_j using the graph-coloring method, see [8]. However, a much simpler decomposition is possible for the toy-problem example considered in this work. Indeed, a uniform finite-element grid has been used to analyze the problem specified in the GFI. For uniform finite-element grids the matrix A is *banded*; furthermore, the number and location of the bands is given by the geometry of the scattering problem. Hence, to decompose the Hamiltonian matrix [Eq. (8)], the simplest way do so is to break it up by band into $m = N_b$ submatrices, with A_j denoting the jth nonzero band of matrix A, and N_b denoting the overall number of its bands. For the square finite-element grid used in the analyzed example, $N_b = 9$. Moreover, because the locations of the bands are known, this decomposition method requires only time of order $O(1)$. Having the matrix decomposition (8), it is then necessary to implement the application of each individual one-sparse Hamiltonian from this decomposition to the actual quantum state of the data register R_2. This "Hamiltonian circuit" can be derived by a procedure resembling the techniques of quantum-random-walk algorithm [30] and is discussed in more detail in Sect. 3.4.5.

After QPEA has been accomplished including the QFT of register R_1, the joined quantum state of registers R_1, R_2 and R_6 becomes, approximately,

$$\begin{aligned} |\Psi\rangle_{1,2,6} = \sum_{j=1}^{N} \Big(& \cos(\phi_b)\tilde{\beta}_j \left|\tilde{\lambda}_j\right\rangle_1 \otimes |u_j\rangle_2 \otimes |0\rangle_6 \\ & + \sin(\phi_b)\beta_j \, |\lambda_j\rangle_1 \otimes |u_j\rangle_2 \otimes |1\rangle_6 \Big), \end{aligned} \tag{9}$$

where λ_j and $|u_j\rangle$ are the eigenvalues and eigenvectors of A, respectively, and $|b\rangle_2 = \sum_{j=1}^{N} \beta_j |u_j\rangle_2$ and $\left|\tilde{b}\right\rangle_2 = \sum_{j=1}^{N} \tilde{\beta}_j |u_j\rangle_2$ are the expansions of quantum states $|b\rangle_2$ and $\left|\tilde{b}\right\rangle_2$, respectively, in terms of these eigenvectors, and $\tilde{\lambda}_j := \lambda_j t_0/2\pi$.

As a *third step*, a further single-qubit ancilla in register R_7 is employed, initially prepared in state $|0\rangle_7$ and then rotated by an angle inversely proportional to the value stored in register R_1, yielding the overall state:

$$\begin{aligned} |\Psi\rangle_{1,2,6,7} = \sum_{j=1}^{N} \Big(& \cos(\phi_b)\tilde{\beta}_j \left|\tilde{\lambda}_j\right\rangle_1 \otimes |u_j\rangle_2 \otimes |0\rangle_6 \\ & + \sin(\phi_b)\beta_j \left|\tilde{\lambda}_j\right\rangle_1 \otimes |u_j\rangle_2 \otimes |1\rangle_6 \Big) \\ & \otimes \left(\sqrt{1 - \frac{C^2}{\lambda_j^2}} \, |0\rangle_7 + \frac{C}{\lambda_j} \, |1\rangle_7 \right), \end{aligned} \tag{10}$$

where $C := 1/\kappa$ is chosen such that $C/\lambda_j < 1$ for all j, because of $\kappa = \lambda_{\max}/\lambda_{\min}$.

Finally, the eigenvalues stored in register R_1 are uncomputed, by the inverse QFT of R_1, inverse Hamiltonian evolution on R_2 and $H^{\otimes n_1}$ on R_1, leaving registers R_1, R_2, R_6, and R_7 in the state

$$\begin{aligned} |\Psi\rangle_{1,2,6,7} \to |0\rangle_1 \otimes \sum_{j=1}^{N} \Big(& \cos(\phi_b)\tilde{\beta}_j \left|u_j\right\rangle_2 \otimes |0\rangle_6 \\ & + \sin(\phi_b)\beta_j \left|u_j\right\rangle_2 \otimes |1\rangle_6 \Big) \\ & \otimes \left(\sqrt{1 - \frac{C^2}{\lambda_j^2}}\, |0\rangle_7 + \frac{C}{\lambda_j} |1\rangle_7 \right). \end{aligned} \tag{11}$$

Ignoring register R_1 and collecting all terms that are not entangled with the term $|1\rangle_6 \otimes |1\rangle_7$ into a "garbage state" $|\Phi_0\rangle_{2,6,7}$, the common quantum state of registers R_2, R_6, and R_7 can be written as, see [5]:

$$\begin{aligned} |\Psi\rangle_{2,6,7} = & (1 - \sin^2(\phi_b)\sin^2(\phi_x))^{1/2} |\Phi_0\rangle_{2,6,7} \\ & + \sin(\phi_b)\sin(\phi_x) |x\rangle_2 \otimes |1\rangle_6 \otimes |1\rangle_7 \ , \end{aligned} \tag{12}$$

where

$$|x\rangle_2 = \frac{1}{\sin\phi_x} \sum_{j=1}^{N} \frac{C\beta_j}{\lambda_j} \left|u_j\right\rangle_2 \tag{13}$$

is the normalized solution to $A|x\rangle = |b\rangle$ stored in quantum data register R_2 and $\sin^2\phi_x := C^2 \sum_{j=1}^{N} |\beta_j|^2/\lambda_j^2$. Note that the solution vector [Eq. (13)] in register R_2 is correlated with the value 1 in the auxiliary register R_7. Hence, if register R_7 is measured and the value 1 is found, we know with certainty that the desired solution of the problem is stored in the quantum amplitudes of the quantum state of register R_2, which can then either be revealed by an ensemble measurement (a statistical process requiring the whole procedure to be run many times) or useful information can also be obtained by computing its overlap $|\langle R|x\rangle|^2$ with a particular (known) state $|R\rangle$ (corresponding to a specific vector $\mathbf{R} \in \mathbb{C}^N$) that has been prepared in a separate quantum register. To avoid nonunitary postselection processes, CJS-QLSA [5] employs QAEA.[11]

With respect to the particular application example that has been analyzed here, namely, solving Maxwell's equations for a scattering problem using the FEM technique, we are interested in the *radar scattering cross section* (RCS) σ_{RCS}, which can be expressed in terms of the modulus squared of a scalar product, $\sigma_{\text{RCS}} = \frac{1}{4\pi}|\mathbf{R}\cdot\mathbf{x}|^2$,

[11] Note that $1/\lambda_{\max} \le \kappa \sin\phi_x \le 1/\lambda_{\min}$, which suggests that $M \sim O(\kappa/\varepsilon)$ would be sufficient to estimate $\alpha_x := \sin^2\phi_x$ with multiplicative error ε, see corollary (6). This is a conservative estimate, and the implied associated cost for the QAE step is indeed by a factor $O(\kappa)$ higher than that assumed by CJS in deriving the overall complexity [Eq. (3)].

where $\mathbf{x}$ is the solution of $A\mathbf{x} = \mathbf{b}$ and $\mathbf{R}$ is an N-dim vector whose components are computed by a 2D surface integral involving the corresponding edge basis vectors and the radar polarization, as outlined in detail in [5]. Thus, to obtain the cross section using QLSA, we must compute $|\langle R|x\rangle|^2$, where $|R\rangle$ is the quantum-theoretic representation of the classical vector $\mathbf{R}$. It is important to note that, whereas $|R\rangle$ and $|x\rangle$ are normalized to 1, the vectors $\mathbf{R}$ and $\mathbf{x}$ are in general not normalized and carry units. Hence, after computing $|\langle R|x\rangle|^2$, units must be restored to obtain $|\mathbf{R}\cdot\mathbf{x}|^2$.

As for $|b\rangle$, the preparation of the quantum state $|R\rangle$ is imperfect. Employing the same preparation procedure that has been used to prepare $|b_T\rangle$, but with oracle R instead of oracle b, we can prepare the entangled state

$$|R_T\rangle_{3,8} = \cos(\phi_r)\left|\tilde{R}\right\rangle_3 \otimes |0\rangle_8 + \sin(\phi_r)\,|R\rangle_3 \otimes |1\rangle_8\,, \tag{14}$$

where the multiqubit quantum *data register* R_3 consisting of $n_3 = \lceil \log_2(2N)\rceil$ qubits contains (as a quantum superposition) the desired arbitrary state $|R\rangle$ entangled with value 1 in an auxiliary single-qubit register R_8, as well as a garbage state $\left|\tilde{R}\right\rangle$ (denoted by the tilde) entangled with value 0 in register R_8. Moreover, the amplitudes squared are given as $\sin^2\phi_r = \frac{C_R^2}{2N}\sum_{j=0}^{2N-1} R_j^2$ and $\cos^2\phi_r = \frac{1}{2N}\sum_{j=0}^{2N-1}\left(1 - C_R^2 R_j^2\right)$, where $C_R = 1/\max(R_j)$, cf. [5]. As outlined in [5], the state (14) is adjoined to state (12) along with a further ancilla qubit in single-qubit register R_9 that has been initialized to state $|0\rangle_9$. Then, a Hadamard gate is applied to the ancilla qubit in register R_9 and a controlled swap operation is performed between registers R_2 and R_3 controlled on the value of the ancilla qubit in register R_9, which finally is followed by a second Hadamard transformation of the ancilla qubit in register R_9. After a few simple classical transformations, the algorithm can compute the scalar product between $|x\rangle$ and $|R\rangle$ as, cf. [5]:

$$|\langle R|x\rangle|^2 = \frac{P_{1110} - P_{1111}}{\sin^2\phi_b \sin^2\phi_x \sin^2\phi_r}, \tag{15}$$

where P_{1110} and P_{1111} denote the probability of measuring a "1" in the three ancilla registers R_6, R_7 and R_8 and a "0" or "1" in ancilla register R_9, respectively. Finally, after restoring the units to the normalized output of QLSA, the RCS in terms of quantities received from the quantum computation is, cf. [5]:

$$\sigma_{\text{RCS}} = \frac{1}{4\pi}\frac{N^2}{C_b^2 C_r^2}\frac{\sin^2\phi_b}{\sin^2\phi_x}(\sin^2\phi_{r0} - \sin^2\phi_{r1}), \tag{16}$$

where $\sin\phi_{r0} := P_{1110}^{\frac{1}{2}}\sin\phi_r$ and $\sin\phi_{r1} := P_{1111}^{\frac{1}{2}}\sin\phi_r$.

It is important to note that, because all the employed state preparation and linear-system-solving operations are unitary, the four amplitudes $\sin\phi_b$, $\sin\phi_x$, $\sin\phi_{r0}$ and $\sin\phi_{r1}$ that are needed for the computation of the RCS according to Eq. (16) can be estimated nearly deterministically (with error ε) using QAEA which allows to avoid

nested nondeterministic subroutines involving postselection.[12] Yet, there is a small probability of failure, which means that QLSA can occasionally output an estimate $\tilde{\sigma}_{\mathrm{RCS}}$ that is *not* within the desired precision range of the actual correct value σ_{RCS}. The failure probability is generally always nonzero but can be made negligible.[13]

3.4 QLSA: algorithm profiling and quantum-circuit implementation

The high-level structure of QLSA [5] is captured by a tree diagram depicted in Fig. 2. It consists of several high-level subroutines hierarchically comprising (i)‘ ‘Amplitude Estimation” (first level), (ii) “State Preparation” and “Solve for x” (second level), (iii) “Hamiltonian Simulation” (third level), and several further sublevel subroutines, such as “HsimKernel” and “Hmag” that are used as part of HS. Figure 2 illustrates the *coarse-grained profiling* of QLSA for the purpose of an accurate LRE of the algorithm, demonstrating the use of repetitive patterns, i.e., templates representing algorithmic building blocks that are reused frequently. Representing each algorithmic building block in terms of a quantum circuit thus yields a step-by-step hierarchical circuit decomposition of the whole algorithm down to elementary quantum gates and measurements. The cost of each algorithmic building block is thereby measured in terms of the number of calls of lower-level subroutines or directly in terms of the number of specified elementary gates, data qubits, ancilla uses, etc.

To obtain an accurate LRE of QLSA, we thus need to represent each algorithmic building block in terms of a quantum circuit that then enables us to count elementary resources. In what follows, we present quantum circuits for selected subroutines of

[12] However, it ought to be noted that, by *“principle of deferred measurements”* (see [1]), for any quantum circuit involving measurements whose results are used to conditionally control subsequent quantum circuits, the actual measurements can always be *deferred* to the very end of the entire quantum algorithm, without in any way affecting the probability distribution of its final outcomes. In other words, measuring qubits commutes with conditioning on their postselected outcomes. Hence, any quantum circuit involving postselection can always be included as a subroutine using only pure states as part of a bigger algorithm with probabilistic outcomes. Nonetheless, in view of the resources used to achieve efficient simulation, measuring qubits as early as possible can potentially reduce the maximum number of *simultaneously* employed physical qubit systems enabling the algorithm to be run on a smaller quantum computer. In addition, we here emphasize that, with a small amount of additional effort, QAEA can be designed such that its final measurement outcomes nearly deterministically yield the desired estimates. Note that a similar concept also applies to QAA in HHL-QLSA, which aims at amplifying the success probability.

[13] The RCS in Eq. (16) is of the form $\sigma_{\mathrm{RCS}} = C\frac{\alpha_1}{\alpha_2}(\alpha_3-\alpha_4)$, where C is a constant and α_i $(i = 1,\ldots,4)$ are the modulī squared of four different quantum amplitudes to be estimated using QAEA. The QAE control register size n_0 has been chosen such (see Table 1) that, with a success probability greater than $1-\varepsilon$, respectively, the corresponding estimates are within $\pm\varepsilon\alpha_i$ of the actual correct values, i.e., $\tilde{\alpha}_i = \alpha_i \pm \varepsilon\alpha_i$. It is straightforward to show that, with only a single run of each of the four QAEA subroutines, our estimate $\tilde{\sigma}_{\mathrm{RCS}} = C\frac{\tilde{\alpha}_1}{\tilde{\alpha}_2}(\tilde{\alpha}_3-\tilde{\alpha}_4)$ for RCS satisfies $\tilde{\sigma}_{\mathrm{RCS}} = \sigma_{\mathrm{RCS}} \pm \varepsilon\sigma_{\mathrm{RCS}} \pm \varepsilon\sigma_{\mathrm{RCS}} \pm \varepsilon\sigma_{\mathrm{RCS}} + O(\varepsilon^2)$, and hence $|\tilde{\sigma}_{\mathrm{RCS}} - \sigma_{\mathrm{RCS}}| \leq 3\varepsilon\sigma_{\mathrm{RCS}}$, with a probability at least $(1-\varepsilon)^4 \approx 1-4\varepsilon$. Note that, to ensure $|\tilde{\sigma}_{\mathrm{RCS}} - \sigma_{\mathrm{RCS}}| \leq \varepsilon\sigma_{\mathrm{RCS}}$ with a probability close to 1, we actually should have chosen an even higher calculation accuracy for each of the four QAEA subroutines, achieved by using the larger QAE control register size $n_0' = \lceil \log_2 M' \rceil$, where $M' = 2^{\lceil \log_2(1/\varepsilon'^2)\rceil}$, enabling estimations with the smaller error $\varepsilon' := \varepsilon/4$. However, we avoided these details in our LRE analysis, which aims at estimating the optimistic resource requirements that are necessary (not imperatively sufficient) to achieve the calculation accuracy $\varepsilon = 0.01$ for the whole algorithm.

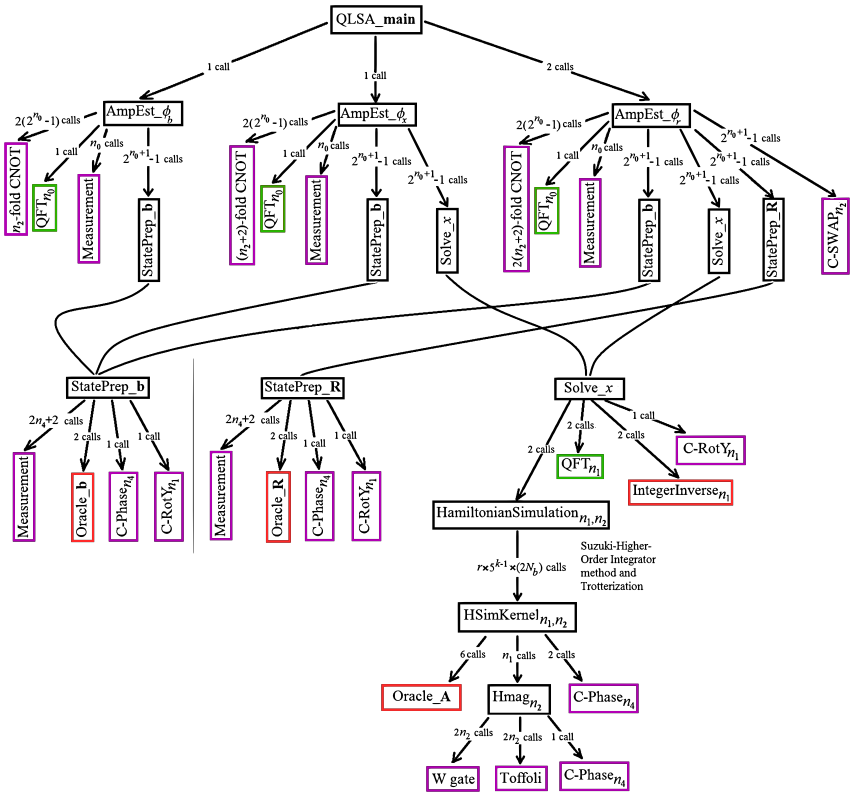

Fig. 2 (Color online) Coarse-grained QLSA profiling overview. The high-level structure of QLSA consists of several high-level subroutines (represented as *black-framed boxes*) hierarchically comprising (i) "Amplitude Estimation" (first level), (ii) "State Preparation" and "Solve for x" (second level), and (iii) "Hamiltonian Simulation" (third level), which includes several further sublevel subroutines, such as "HsimKernel" and "Hmag." These subroutines are further "partitioned" into more fine-grained repetitive algorithmic building blocks (such as, e.g., QFT, oracle query implementations, multicontrolled NOTs and multicontrolled rotations, etc.) that are eventually hierarchically decomposed down to elementary quantum gates and measurements. Among them, well-known library functions, such as QFT, are shown as *green-framed boxes*; single-qubit measurements (in computational basis); and well-established composite gates and multiqubit-controlled gates (such as Toffoli, W-gate and multicontrolled NOTs) are represented by *purple-framed boxes*; automated implementations of oracles and the "IntegerInverse" subroutine are illustrated as *red-framed boxes*. For multiqubit gates, the number of qubits involved is indicated by a subscript or a prefix label; for example, a QFT acting on n_0 qubits is represented as "QFT$_{n_0}$"; a multicontrolled NOT employing n_2 control quits is denoted as "n_2-fold CNOT." The number of calls of each algorithmic building block is indicated by a *labeled arrow*. The cost of a subroutine is measured in terms of the number of specified gates, data qubits, ancilla uses, etc., or/and in terms of calls of lower-level subsubroutines and their associated costs. Note that the cost may vary depending on input argument value to the subroutine. To obtain the LRE of the whole algorithm, multiply the number of calls of each lowest-level subroutine with its elementary resource requirement. The cost of the lowest-level subroutines and oracles is provided in the form of tables in the "Appendix." It also becomes apparent how the overall run-time of QLSA accrues through a series of nested loops consisting of numerous iterative steps that dominate the run-time and others whose contributions are insignificant and can be neglected. The dominant contributions to run-time are given by those paths within the tree diagram which include Hamiltonian Simulation as the most resource-demanding bottleneck, involving Trotterization with $r \approx 10^{12}$ time-splitting slices, with each Trotter slice involving iterating over each matrix band to implement the corresponding part of Hamiltonian state transformation, which (for each band) requires several oracle A implementations to compute the matrix elements

QLSA. Well-known circuit decompositions of common multiqubit gates (such as, e.g., Toffoli gate, multicontrolled NOTs, and W gate) and their associated resource requirements are discussed in the "Appendix."

3.4.1 The "main" function QLSA_main

The task of the main algorithm "QLSA_main" is to estimate the radar cross section for a FEM scattering problem specified in GFI using the quantum amplitude estimation subalgorithms "AmpEst_ϕ_b," "AmpEst_ϕ_x" and "AmpEst_ϕ_r" to approximately compute the angles corresponding to the probability amplitudes $\sin(\phi_b)$, $\sin(\phi_x)$, $\sin(\phi_{r0})$ and $\sin(\phi_{r1})$:

$$
\begin{aligned}
\phi_b &\leftarrow \text{AmpEst\_}\phi_b(\text{Oracle\_}\mathbf{b}) \\
\phi_x &\leftarrow \text{AmpEst\_}\phi_x(\text{Oracle\_}\mathbf{A}, \text{Oracle\_}\mathbf{b}) \\
\phi_{r0} &\leftarrow \text{AmpEst\_}\phi_r(\text{Oracle\_}\mathbf{A}, \text{Oracle\_}\mathbf{b}, \text{Oracle\_}\mathbf{R}, 0) \\
\phi_{r1} &\leftarrow \text{AmpEst\_}\phi_r(\text{Oracle\_}\mathbf{A}, \text{Oracle\_}\mathbf{b}, \text{Oracle\_}\mathbf{R}, 1)
\end{aligned}
$$

where in the last two lines "0" and "1" refer to the probability of measuring value 0 or 1 on ancilla qubit in register R_9, respectively. It then uses these probability amplitudes (or rather their corresponding probabilities) to calculate an estimate of the radar cross section $\sigma_{\text{RCS}} = \sigma_{\text{RCS}}(\phi_b, \phi_x, \phi_{r0}, \phi_{r1})$ according to Eq. (16), whereby this part uses only classical computation. The result of the whole computation ought to be as precise as specified by the multiplicative error term $\pm\varepsilon\sigma_{\text{RCS}}$, where the desired (given) accuracy parameter in our analysis has the value $\varepsilon = 0.01$. The LRE of the complete QLS algorithm is thus obtained as the sum of the LREs of the four calls of the quantum amplitude estimation subalgorithms, respectively, that are employed by QLSA_**main**.

3.4.2 Amplitude estimation subroutines

In this subsection we present the quantum circuits of the three *Amplitude Estimation* subroutines "AmpEst_ϕ_b," "AmpEst_ϕ_x" and "AmpEst_ϕ_r," which are called by "QLSA_**main**" to compute estimates of the angles ϕ_b, ϕ_x, ϕ_{r0} and ϕ_{r1} that are needed to obtain an estimate for the RCS σ_{RCS}.

Subroutine AmpEst_ϕ_b This subroutine computes an estimate for the angle ϕ_b, which determines the probability amplitude of success $\sin(\phi_b)$ for the preparation of the quantum state $|b\rangle$ in register R_2, see Eq. (7). Its algorithmic structure is represented by the circuits depicted in Figs. 3, 4 and 5. It employs subroutine "StatePrep_**b**," which prepares the state [Eq. (7)], and a *Grover iterator* whose construction is illustrated by the circuit in Fig. 5.

Subroutine AmpEst_ϕ_x This subroutine computes an estimate for the angle ϕ_x, which, together with the previously computed angle ϕ_b, determines the probability amplitude of success, $\sin(\phi_b)\sin(\phi_x)$, of computing the solution state $|x\rangle$ in register R_2, see Eq. (12). Its algorithmic structure is represented by the circuits depicted in Figs. 6, 7 and 8. It involves subroutine "StatePrep_**b**," which prepares the quantum state (7), the subroutine "Solve_x," which implements the actual "solve-for-x" procedure that incorporates all required lower-level subroutines such as those needed for Hamiltonian Simulation, and a *Grover iterator* whose construction is given in Fig. 8.

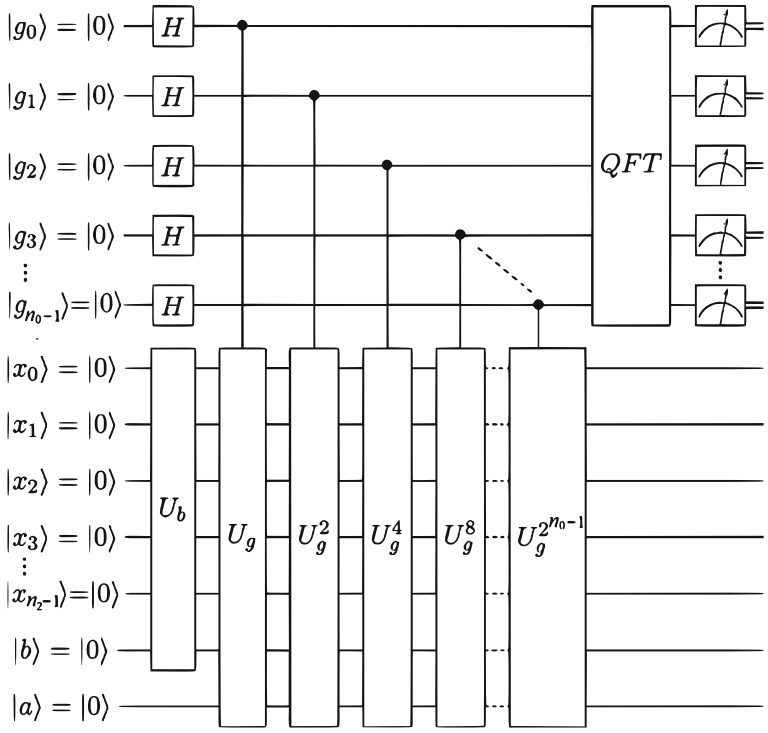

Fig. 3 Circuit to implement subroutine "AmpEst_ϕ_b," which computes an estimate for angle ϕ_b. The unitary transformations U_b and U_g are explained in Figs. 4 and 5. The amplitude estimation subroutine is completed by a QFT of the QAE control register R_0 (here represented by wires $|g_0\rangle, \ldots, |g_{n_0-1}\rangle$) and measuring it in the computational basis. The measurement result $\mathbf{g} = (g[0], \ldots, g[n_0 - 1])$ is recorded, $y \leftarrow \mathbf{g}$, and used to compute the estimate $\phi_b = (\pi y/M)$, cf. [22]

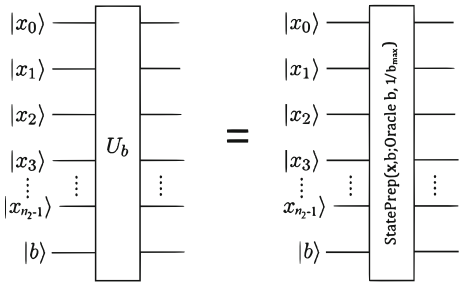

Fig. 4 Unitary transformation U_b is an abbreviation for subroutine "StatePrep_**b**," whose circuit representation is discussed in Sect. 3.4.3

Subroutine AmpEst_ϕ_r This subroutine computes an estimate for the angle ϕ_{r0} or ϕ_{r1}, respectively, which, together with the previously computed angles ϕ_b and ϕ_x, determine the probability amplitude of successfully computing the overlap integral $\langle R|x\rangle$. Its algorithmic structure is represented by the circuits depicted in Figs. 9, 10, 11 and 12. It involves subroutines "StatePrep_**b**" and "StatePrep_**R**," which prepare the quantum states (7) and (14), respectively, the subroutine "Solve_x," which implements the actual "solve-for-x" procedure, and furthermore a swapp protocol that is required

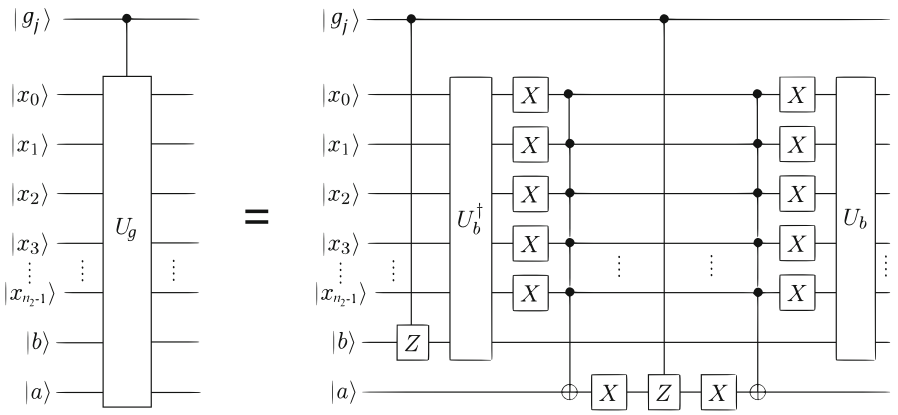

Fig. 5 Quantum circuit of the (unitary) *Grover iterator* U_g employed by subroutine AmpEst_ϕ_b; its action is to be controlled by control-register qubit $g[j]$

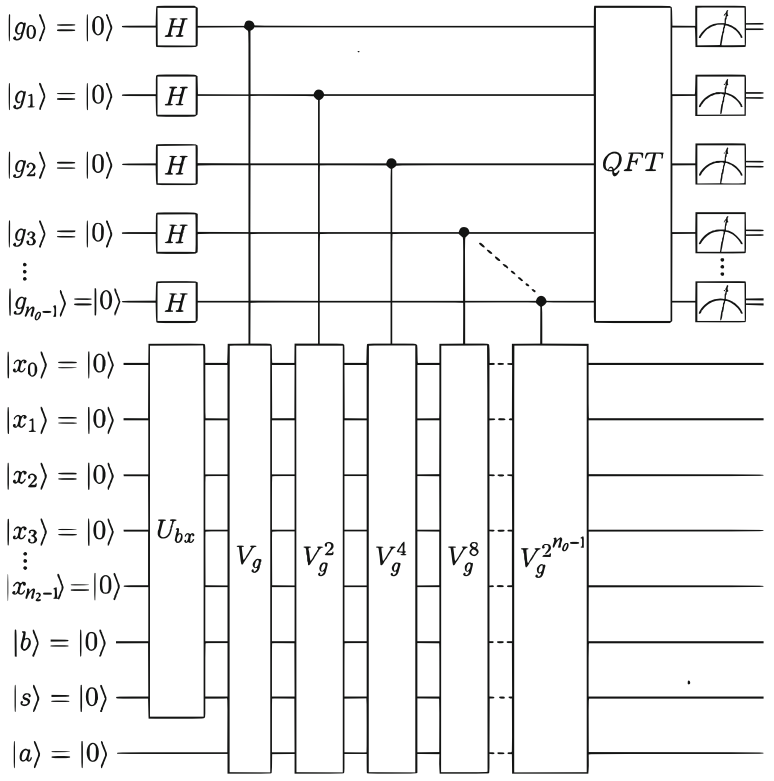

Fig. 6 Circuit to implement subroutine "AmpEst_ϕ_x," which computes an estimate for angle ϕ_x. The unitary transformations U_{bx} and V_g are explained in Figs. 7 and 8. The amplitude estimation subroutine is completed by a QFT of the QAE control register R_0 (here represented by wires $|g_0\rangle, \ldots, |g_{n_0-1}\rangle$) and measuring it in the computational basis. The measurement outcome $\mathbf{g} = (g[0], \ldots, g[n_0 - 1])$ is recorded, $y \leftarrow \mathbf{g}$, and used to compute the estimate $\phi_x = (\pi y/M)$, cf. [22]

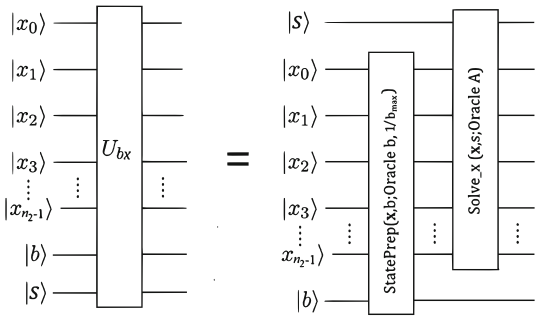

Fig. 7 Unitary transformation U_{bx} consists of two subroutines: "StatePrep_**b**" followed by "Solve_x," whose circuit representations are discussed in Sects. 3.4.3 and 3.4.4

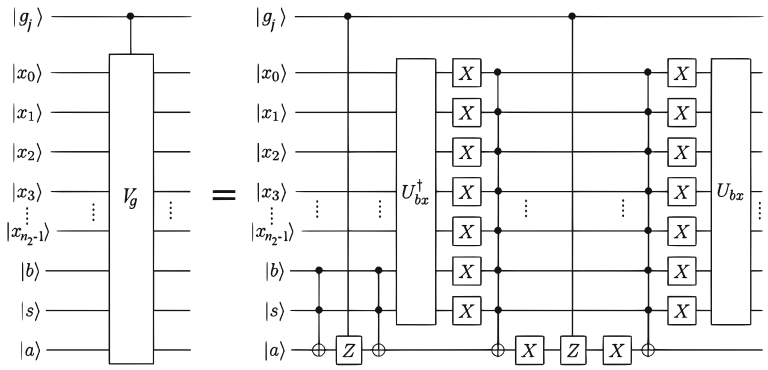

Fig. 8 Quantum circuit of the (unitary) *Grover iterator* V_g employed by subroutine "AmpEst_ϕ_x"; its action is to be controlled by control-register qubit $g[j]$

for computing an estimate of $\langle R|x\rangle$, and finally a *Grover iterator* whose construction is given by the quantum circuit in Fig. 12.

3.4.3 State preparation subroutine

The *state preparation* subroutine "StatePrep" is used to generate the quantum states $|b_T\rangle$ and $|R_T\rangle$ in Eqs. (7) and (14) from given classical vectors **b** and **R** using the corresponding oracles and controlled-phase and rotation gates. The circuit for generating $|b_T\rangle$ is depicted in Fig. 13. A similar circuit is used to generate $|R_T\rangle$, by replacing the Oracle b by Oracle R. The subroutines "C-Phase" and "C-RotY" and their associated resource counts are discussed in Appendix "Controlled phase: C-Phase(**c**; ϕ_0, f)" and "Controlled-RotY: C-RotY(**c**, **t**; ϕ_0, f)," respectively. The implementation of Oracles b and R is analyzed in Sect. 4.

*3.4.4 Solve_*x *subroutine*

Subroutine "Solve_x(**x**, **s**; Oracle_A)" is the actual linear-system-solving procedure, i.e., it implements the "solve-for-x") transformation. More concretely, it takes as input the state $|b_T\rangle_{2,6}$ (see Eq. (7)) that has been prepared in registers R_2, R_6, and computes

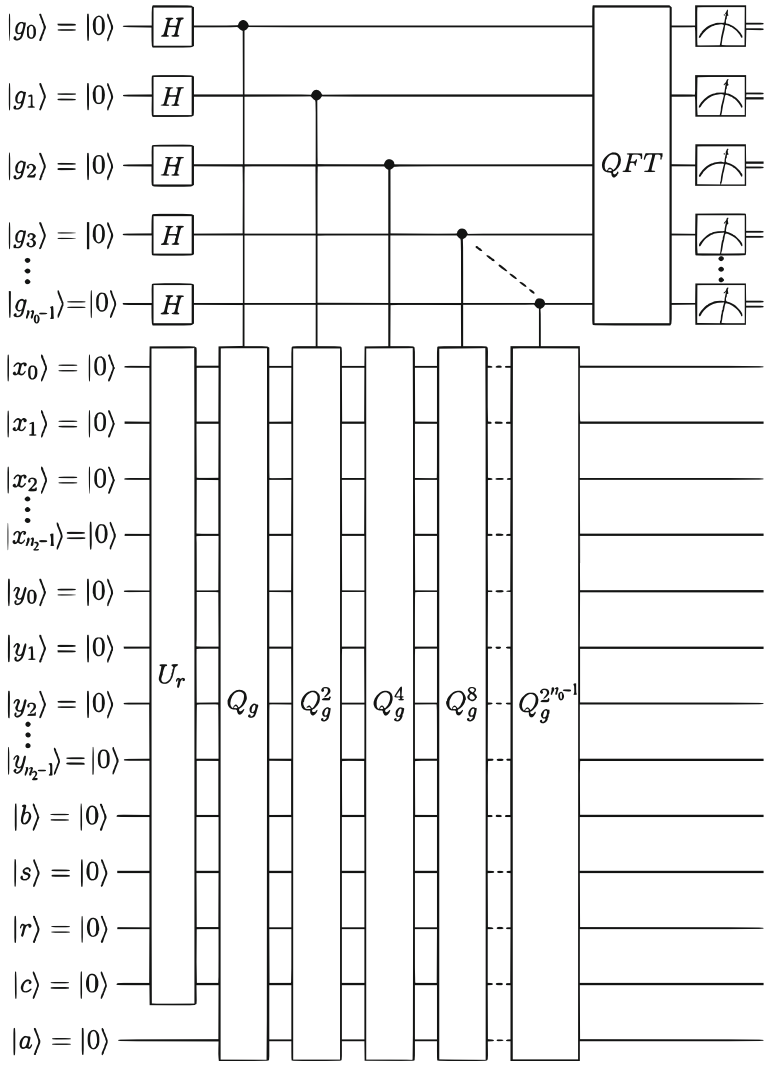

Fig. 9 Quantum circuit to implement subroutine "AmpEst_ϕ_r," which computes an estimate for the angle ϕ_{r0} or ϕ_{r1}, respectively, which, together with the previously computed angles ϕ_b and ϕ_x, are needed to calculate an estimate of RCS according to Eq. (16). The unitary transformations U_r and Q_g are explained in Figs. 10 and 12. The amplitude estimation subroutine is completed by a QFT of the QAE control register R_0 (represented by wires $|g_0\rangle, \ldots, |g_{n_0-1}\rangle$) and measuring it in the computational basis. The measurement result $\mathbf{g} = (g[0], \ldots, g[n_0 - 1])$ is recorded, $y \leftarrow \mathbf{g}$, and used to compute the estimate $\phi_{rf} = (\pi y/M)$, cf. [22], depending on the value of the flag $f \in \{0, 1\}$ used by unitary Q_g, see Fig. 12

the state given in Eq. (12) which contains the solution state $|x\rangle_2 = A^{-1}|b\rangle_2$ in register R_2 with success-probability amplitude $\sin(\phi_b)\sin(\phi_x)$. The arguments of this subroutine are **x** and **s** corresponding to the input states in data register R_2 and single-qubit ancilla register R_7; furthermore, Oracle_A occurs in the argument list to indicate

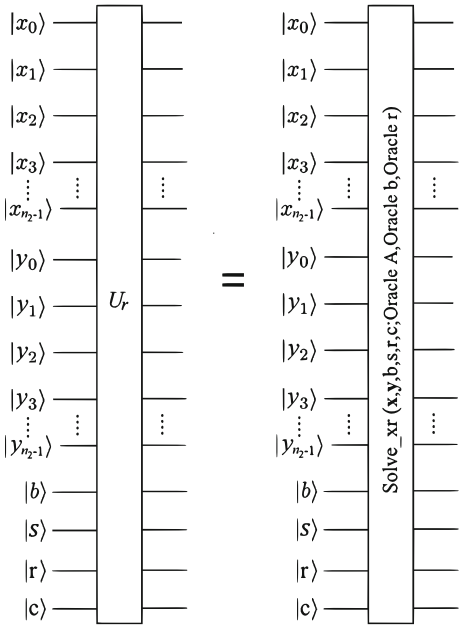

Fig. 10 Unitary transformation U_r is an abbreviation for the subroutine "Solve_xr," whose circuit representation is provided in Fig. 11

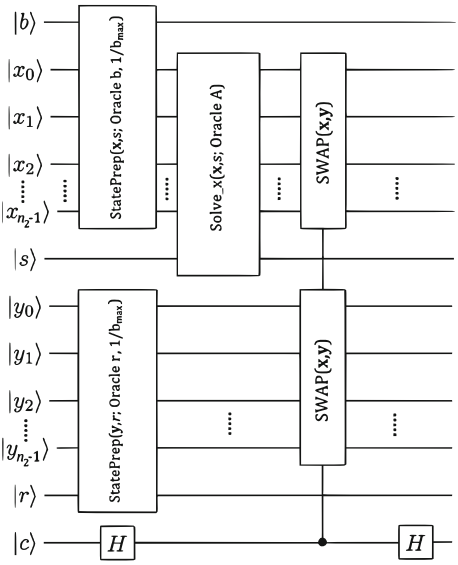

Fig. 11 Definition of subroutine "Solve_xr" that is shown in Fig. 10 to define the unitary transformation U_r. This subroutine starts with implementing the preparation of quantum states (7) and (14) in registers R_2, R_6 and R_3, R_8 (here given as $|x_0\rangle, \ldots, |x_{n_2-1}\rangle$, $|b\rangle$ and $|y_0\rangle, \ldots, |y_{n_2-1}\rangle$, $|r\rangle$), respectively; then it employs subroutine "Solve_x," which implements the actual "solve-for-x" procedure; finally, a Hadamard gate is applied to the ancilla qubit in register R_9 (here labeled as $|c\rangle$) and a controlled swap protocol is performed between registers R_2 and R_3 controlled on the value of the ancilla qubit in register R_9, which finally is followed by a second Hadamard gate on the ancilla qubit in register R_9. The swap protocol is required for computing an estimate of the overlap $\langle R|x\rangle$

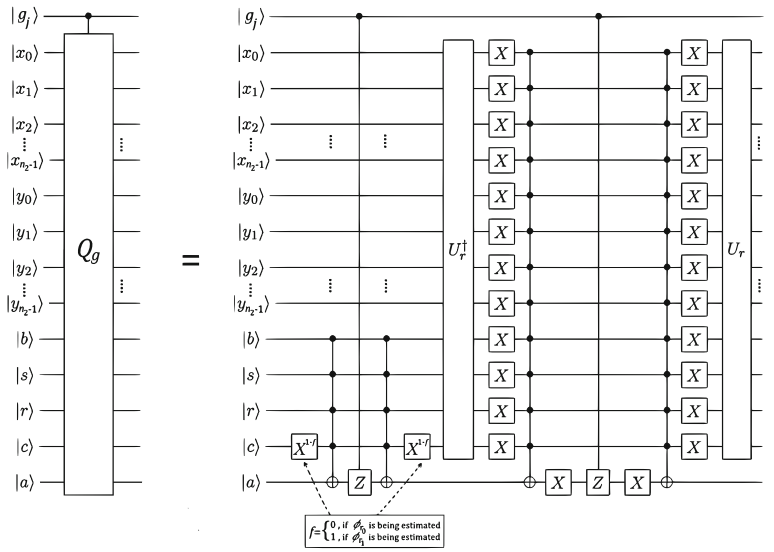

Fig. 12 Quantum circuit of the (unitary) *Grover iterator* Q_g employed by subroutine "AmpEst_ϕ_r"; its action is to be controlled by control-register qubit $g[j]$. The value of the flag $f \in \{0, 1\}$ determines whether the angle ϕ_{r0} or ϕ_{r1} is to be estimated, respectively

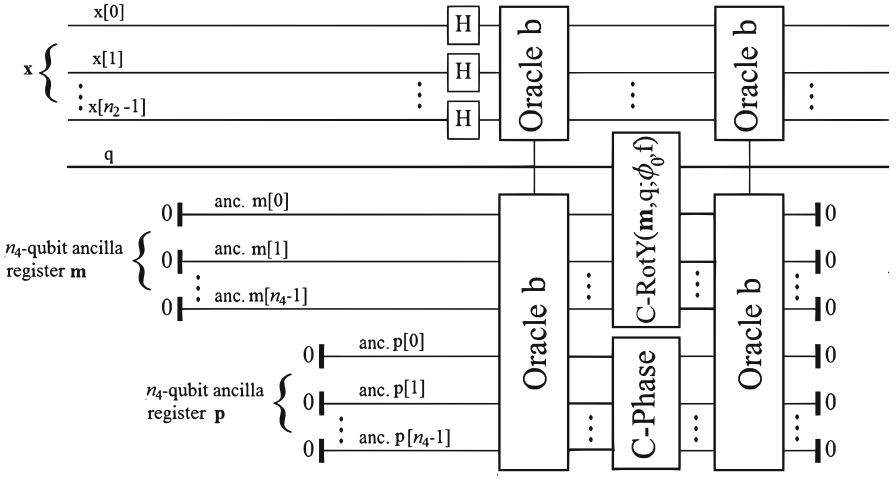

Fig. 13 Quantum circuit to implement the subroutine "StatePrep(**x**, q; Oracle **b**, $1/b_{\max}$)," which generates the quantum state $|b_T\rangle_{2,6}$ in Eq. (7). In addition to the data register R_2 (function argument **x**; here represented by wires $x[0], \ldots, x[n_2-1]$) and single-qubit ancilla register R_6 (here represented by wire q), the procedure involves two further, auxiliary computational registers R_4 and R_5, each consisting of n_4 ancilla qubits (here represented by wires $m[0], \ldots, m[n_4-1]$ and $p[0], \ldots, p[n_4-1]$), respectively. The latter two registers **m** and **p** are used to store the magnitude and phase components, b_j and ϕ_j, respectively. Following the Oracle b queries, a controlled-phase gate is applied to the auxiliary single-qubit register **q**, controlled by the calculated value of the phase carried by n_4-qubit ancilla register **p**; in addition, the single-qubit register **q** is rotated conditioned on the calculated value of the amplitude (magnitude) carried by the n_4-qubit ancilla register **m**. Uncomputing registers **m** and **p** involves further oracle b calls. The subroutine "StatePrep(**y**, r; Oracle **r**, $1/R_{\max}$)" generating the quantum state $|R_T\rangle$ is implemented by a similar circuit, with Oracle r instead of Oracle b

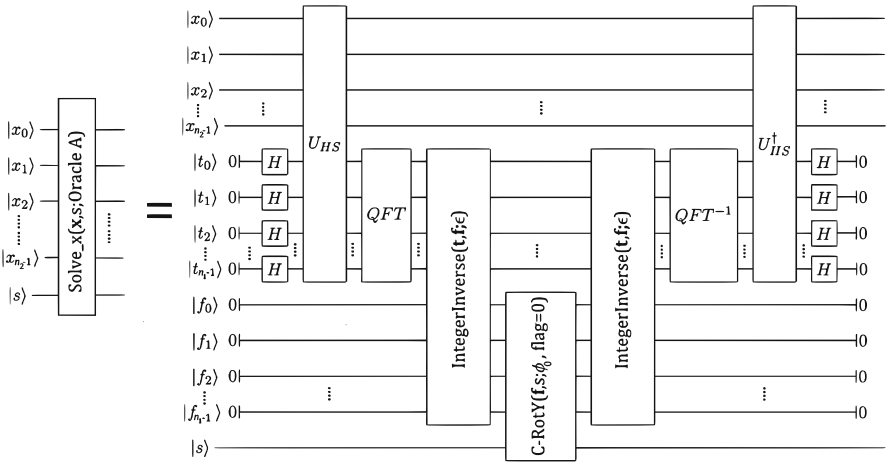

Fig. 14 Quantum circuit to implement subroutine "Solve_x($\mathbf{x}$, $\mathbf{s}$; Oracle$A$)." Register R_2 (here represented by wires labeled as $|x_0\rangle, \ldots, |x_{n_2-1}\rangle$) carries the input state $|b_T\rangle_{2,6}$ defined in Eq. (7); the register R_6 is ignored here, as Solve_x does not act on the latter. The output state of Solve_x is stored in register R_2; it contains the solution $|x\rangle_2 = A^{-1}|b\rangle_2$ with success-probability amplitude $\sin(\phi_b)\sin(\phi_x)$ (see Eq. (12). Quantum register R_1 (here represented by wires $|t_0\rangle, \ldots, |t_{n_1-1}\rangle$) is the *control register* for the HS procedure, which is represented by the unitary transformation U_{HS} that is defined in Fig. 15 and elaborated on below. U_{HS} and its Hermitian conjugate $U_{HS}^{\dagger}$ act on register R_2, with the action being controlled by $|\mathbf{t}\rangle_{R_1}$ that has been initialized to state $|\phi\rangle_1 := H^{\otimes n_1}|0\rangle^{\otimes n_1}$. Following U_{HS}, QFT is performed on register R_1 to complete the implementation of QPEA and so acquire information about the eigenvalues of A and store them in register R_1. A local auxiliary n_1-qubit register R_{11} is employed (here represented by wires $|f_0\rangle, \ldots, |f_{n_1-1}\rangle$) that has been initialized to state $|\mathbf{0}\rangle_{11} \equiv |0\rangle^{\otimes n_1}$. By subroutine "IntegerInverse":$|\mathbf{t}\rangle_1 \otimes |\mathbf{0}\rangle_{11} \rightarrow |\mathbf{t}\rangle_1 \otimes |\mathbf{1/t}\rangle_{11}$, whose implementation is discussed in Sect. 4, ancilla register R_{11} obtains the inverse value λ_j^{-1} of the eigenvalue λ_j stored in HS control register R_1. Next, the controlled rotation "C-RotY" (see Appendix "Controlled-RotY: C-RotY($\mathbf{c}$, $\mathbf{t}$; ϕ_0, f)" for details) rotates the quantum state of single-qubit register R_7 (here labeled as $|s\rangle$) by an angle proportional to the value stored in register R_{11}, i.e., inversely proportional to the eigenvalue stored in register R_1; this step implements the transformation yielding the quantum state in Eq. (10). Finally, registers R_1 and R_{11} are uncomputed and terminated by the inverse operation of IntegerInverse on R_1 and R_{11}, inverse QFT of R_1, inverse Hamiltonian evolution of R_2, applying $H^{\otimes n_1}$ on R_1 and measuring the value "0" in all corresponding qubits; this step yields the common quantum state (11) for registers R_1, R_2, R_6, and R_7

that it is called by Solve_x to implement the HS lower-level subroutines. Note that "Solve_x" does not act on register R_6.

The quantum circuit for "Solve_x" is shown in Fig. 14. It involves lower-level subroutines "HamiltonianSimulation" (see Fig. 15), QFT, "IntegerInverse," and their Hermitian conjugates, respectively, and the controlled rotation "C-RotY," which is defined and analyzed in Appendix "Controlled-RotY: C-RotY($\mathbf{c}$, $\mathbf{t}$; ϕ_0, f)."

3.4.5 Hamiltonian Simulation subroutines

Hamiltonian Simulation subroutines implement, as part of QPEA, the unitary transformation $\exp(-iA\tau t_0/T)$, which is to be applied to register R_2, which together with register R_6 has been prepared in quantum state $|b_T\rangle_{2,6}$, whereby this Hamiltonian evo-

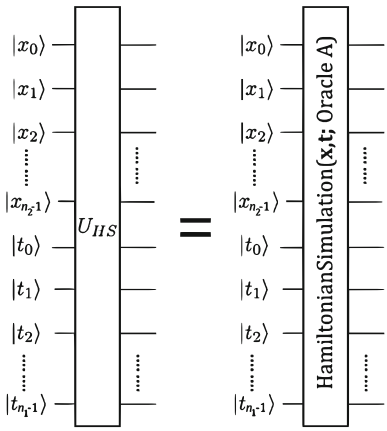

Fig. 15 Unitary transformation U_{HS} is an abbreviation for the "HamiltonianSimulation($\mathbf{x}$, $\mathbf{t}$; Oracle$A$)" subroutine, whose quantum-circuit implementation is given below

lution is to be controlled by HS control register R_1 and the Hamiltonian is specified by Oracle A.

For a thorough HS analysis, see [8] and further references therein. The decomposition of the banded matrix A by band into a sum of submatrices, according to Eq. (8), and the Suzuki-Higher-Order Integrator method [26] with order $k = 2$ and Trotterization [25] are all accomplished by subroutine "HamiltonianSimulation($\mathbf{x}$, $\mathbf{t}$; Oracle$A$)," whose implementation is illustrated in Figs. 16 and 17. The Suzuki-Trotter time-splitting factor, here denoted by r, can be determined by the formula, cf. [8]:

$$r = \lceil 5^{k-1/2}(2N_b\|A\|t)^{1+1/2k}/\varepsilon^{1/2k}\rceil, \tag{17}$$

where $t = \tau t_0/T \leq t_0$ is the length of time the Hamiltonian evolution must be simulated, and $\|A\|$ is the norm of the Hamiltonian matrix. As was shown in [3], to ensure algorithmic accuracy up to error bound ε for subalgorithm "Solve_x," we must have $t_0 \sim O(\kappa/\varepsilon)$. In our analysis, the time constant for Hamiltonian Simulation was set $t_0 = 7\kappa/\varepsilon$, as suggested by the problem specification in the IARPA GFI. Inserting the values $k = 2, N_b = 9, \varepsilon = 0.01$ and $\|A\|t \lesssim 7 \times 10^6$ into Eq. (17) yields the approximate value $r \lesssim 8 \times 10^{11}$. However, to ensure accuracy ε not only for the Hamiltonian-evolution simulation but also for each of the three Amplitude Estimation subroutines that employ subalgorithm "Solve_x" in $(2^{n_0+1} - 1)$ calls, respectively, see Fig. 2, we would typically require a much smaller target accuracy for the implementation of the Hamiltonian evolution. Assuming errors always adding up, an obvious choice would be $\varepsilon' = \varepsilon/(2^{n_0+1} - 1)$, which, when inserted into Eq. (17) in place of ε, yields $r \approx 6.35 \times 10^{12}$. This is a fairly conservative and unnecessarily large estimate, though. Following the suggestions in the GFI, for the purpose of our LRE analysis, we have used the somewhat smaller (average) value $r = 2.5 \times 10^{12}$, which is roughly obtained by using the average Hamiltonian-evolution time $t_0/2$ rather than the maximum HS time t_0 in Eq. (17).

Furthermore, the application of a controlled one-sparse Hamiltonian transformation to any arbitrary input state in register R_2 uses techniques resembling a generalization of the quantum-random-walk algorithm [30]. Its implementation is the task

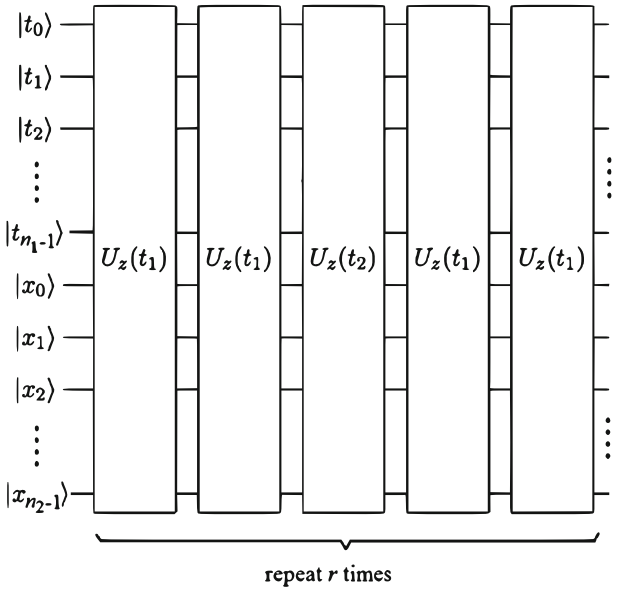

Fig. 16 Quantum circuit to implement subroutine "HamiltonianSimulation($\mathbf{x}$, $\mathbf{t}$; Oracle$A$)" which uses HS control register R_1 (function argument $\mathbf{t}$; represented by wires labeled as $|t_0\rangle, \ldots, |t_{n_1-1}\rangle$) to apply a Hamiltonian transformation of register R_2 (function argument $\mathbf{x}$; represented by wires labeled as $|x_0\rangle, \ldots, |x_{n_2-1}\rangle$), with the Hamiltonian specified by Oracle A. This subroutine comprises the Suzuki-Higher-Order Integrator method with order $k = 2$ and Trotter time-splitting factor r; the value $r = 2.5 \times 10^{12}$ has been used for our LRE. The unitary transformations $U_z(t_1)$ and $U_z(t_2)$ are defined in Fig. 17

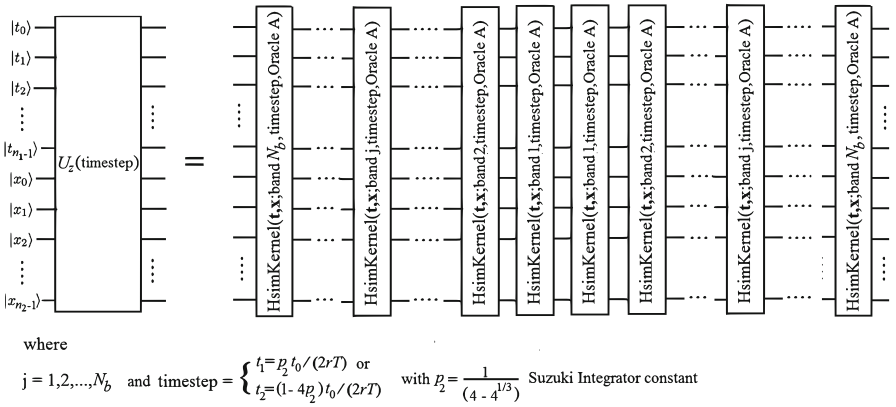

Fig. 17 Definition of the unitary transformation "U_z(timestep)" for two different timesteps timestep $\in \{t_1, t_2\}$, which are determined by Suzuki-Integrator constant p_2 and Trotter time-splitting factor r. $U_z(t_1)$ and $U_z(t_2)$ are used to implement the Suzuki-Higher-Order Integrator [26] as part of the task of the higher-level subroutine "HamiltonianSimulation($\mathbf{x}$, $\mathbf{t}$; Oracle$A$)," see Fig. 16. The implementation of the lower-level subroutine "HsimKernel($\mathbf{t}$, $\mathbf{x}$, band, timestep, Oracle$A$)" is presented in Fig. 18

of the two lower-level subroutines "HsimKernel($\mathbf{t}$, $\mathbf{x}$, band, timestep, Oracle$A$)" and "Hmag($\mathbf{x}$, $\mathbf{y}$, m, ϕ_0)," which are represented and illustrated by circuits in Figs. 18 and 19, respectively.

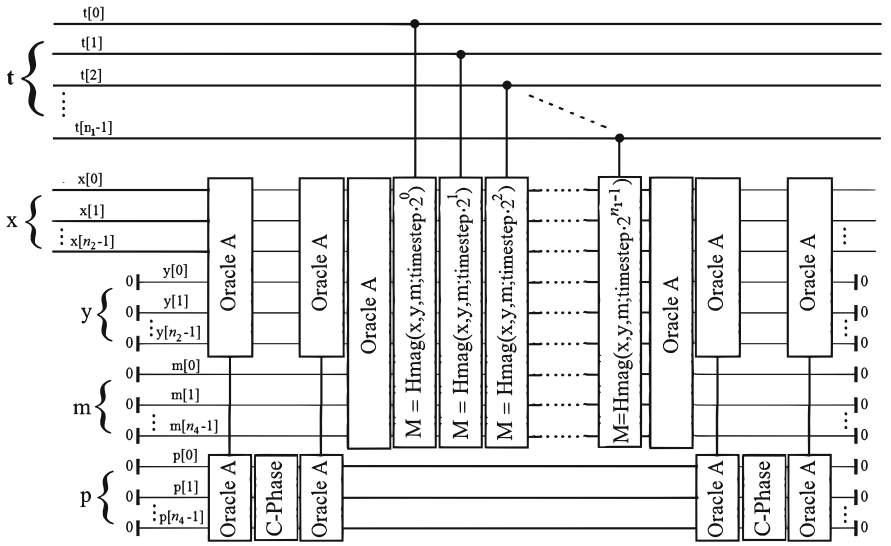

Fig. 18 Quantum circuit to implement the subroutine "HsimKernel(**t**, **x**, band, timestep, Oracle$A$)," whose task is to apply a 1-sparse Hamiltonian to the input state in register R_2 (function argument **x**; here represented by wires $x[0], \ldots, x[n_2 - 1]$), whereby the Hamiltonian transformation is to be controlled by HS control register R_1 (function argument **t**; represented by wires $t[0], \ldots, t[n_1 - 1]$) and Oracle A is used to specify the Hamiltonian. The argument "band" is an *integer* to denote the Hamiltonian band that is to be applied. The argument "timestep" is a *real* timescale factor, which can have two values timestep $\in \{t_1, t_2\}$ (see Fig. 17). Oracle A is a function "Oracle_A(**x**, **y**, **z**; band, argflag)" that accesses Hamiltonian bands and, depending on the value of the integer flag argflag $\in \{0, 1\}$, computes the corresponding magnitude or phase value, respectively, and stores them in an n_4-qubit register $\mathbf{z} \in \{\mathbf{m}, \mathbf{p}\}$. Here, **y** is an n_2-qubit ancilla register R_{12} to hold the connected Hamiltonian node index, and the auxiliary n_4-qubit registers **m** and **p** are used to store the Hamiltonian magnitude and phase value, respectively. These ancilla registers are initialized and terminated to states $|0\rangle^{\otimes n_2}$ and $|0\rangle^{\otimes n_4}$, respectively. The controlled subroutine M :=Hmag(**x**, **y**, m, ϕ_0) is defined in Fig. 19, and the controlled subroutine "C-Phase(**c**; ϕ_0, f)" is discussed in Appendix "Controlled phase: C-Phase(**c**; ϕ_0, f)" and illustrated in Fig. 38

3.4.6 Oracle subroutines

A quantum oracle is commonly considered a unitary "black box" labeled as U_f which, given the value **x** of an n-qubit input register $\mathscr{R}_1$, efficiently and unitarily computes the value of a function $f : \{0, 1\}^n \to \{0, 1\}^m$ and stores it in an m-qubit auxiliary register $\mathscr{R}_2$ that has initially been prepared in state $|0\rangle^{\otimes m}$:

$$U_f : \ |\mathbf{x}\rangle_1 \otimes |\mathbf{0}\rangle_2 \to |\mathbf{x}\rangle_1 \otimes |f(\mathbf{x})\rangle_2 \,. \tag{18}$$

In our analysis, oracles must be employed for the purpose of state preparation (Oracle **b** or Oracle **R**) and Hamiltonian Simulation (Oracle A); they need to be constructed from mappings between the FEM global edge indices and the quantities defining the linear system, matrix A and vector **b**, as well as the "measurement vector" **R** that is used to compute the RCS.

Theoretically, oracle implementations are usually not specified. The efficiency of oracular algorithms is commonly characterized in terms of their query complexity,

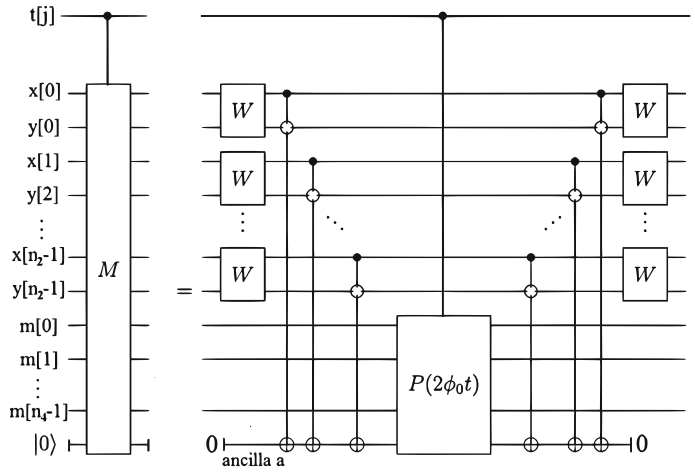

Fig. 19 Quantum circuit to implement the subroutine $M :=$Hmag($\mathbf{x}$, $\mathbf{y}$, m, ϕ_0), whose application is to be controlled by a single-qubit $\mathbf{t}[j]$ that is part of the n_1-qubit HS control register $\mathbf{t}$. Its task is to apply the coupling elements' magnitude component of a 1-sparse Hamiltonian operation; the circuit implementation resembles a generalized quantum walk. Here, $\mathbf{x}$ and $\mathbf{y}$ are n_2-qubit index registers (represented by wires $x[0], \ldots, x[n_2 - 1]$ and $y[0], \ldots, y[n_2 - 1]$), respectively, and $\mathbf{m}$ is an n_4-qubit register (represented by wires $m[0], \ldots, m[n_4 - 1]$), which holds the Hamiltonian magnitude value. The angle ϕ_0 denotes the minimum resolvable phase shift. The W gate and the controlled-phase gate $P(2\phi_0 t)$ are specified in Appendix "W-gate" and "Controlled phase: C-Phase($\mathbf{c}$; ϕ_0, f)," respectively

assuming each query is given by an efficiently computable function. However, in practice oracle implementations must be accounted for. Our analysis aims at comprising all resources, including those which are needed to implement the required oracles. Their automated implementation using the programming language Quipper and its compiler is elaborated on in Sect. 4. Here we briefly discuss the high-level tasks of these oracle functions. Their resource estimates are presented in "Appendix 3."

Oracle **b** is used to prepare quantum state $|b_T\rangle_{2,6}$, see Eq. (7) and Fig. 13. Its task is accomplished by subroutine "Oracle_**b**($\mathbf{x}$, $\mathbf{m}$, $\mathbf{p}$)," which takes as input the quantum state of the n_2-qubit register R_2 (argument $\mathbf{x}$; spanning the linear-system global edge indices), computes the corresponding magnitude value b_j and phase value ϕ_j, and stores them in the two auxiliary computational registers R_4 and R_5 (labeled by arguments $\mathbf{m}$ and $\mathbf{p}$), each consisting of n_4 ancilla qubits and initialized (and later terminated) to states $|0\rangle^{\otimes n_4}$, respectively.

Oracle **R** is used to prepare quantum state $|R_T\rangle_{3,8}$ in Eq. (14). Its task is accomplished by subroutine "Oracle_**R**($\mathbf{x}$, $\mathbf{m}$, $\mathbf{p}$)" which takes as input the quantum state of the n_2-qubit register R_3 (argument $\mathbf{x}$; spanning the FEM global edge indices), computes the corresponding magnitude value r_j and phase value $\phi_j^{(r)}$, and stores them in the two n_4-qubit auxiliary computational registers R_4 and R_5, (labeled by arguments $\mathbf{m}$ and $\mathbf{p}$), each initialized (and later terminated) to states $|0\rangle^{\otimes n_4}$, respectively.

Oracle A is needed to compute the matrix A of the linear system; it is employed as part of the HS subroutine "HsimKernel" to specify the 1-sparse Hamiltonian that is to be applied. This high-level task is accomplished by the "Oracle_A($\mathbf{x}$, $\mathbf{y}$, $\mathbf{z}$; band, argflag)" subroutine, which takes as input the quantum state

of the n_2-qubit register R_2 (argument **x**; spanning the linear-system global edge indices) and returns the connected Hamiltonian node index storing it in an n_2-qubit ancilla register R_{12} (labeled by argument **y**); furthermore, it accesses Hamiltonian bands through the *integer* argument "band" and, depending on the value of the integer variable argflag $\in \{0, 1\}$, computes the corresponding Hamiltonian magnitude or phase value, respectively, and stores it in the corresponding auxiliary n_4-qubit register $\mathbf{z} \in \{\mathbf{m}, \mathbf{p}\}$.

4 Automated resource analysis of oracles via the programming language Quipper

The logical circuits required to implement the Oracles A, b, and R were generated using the quantum programming language Quipper and its compiler. Quipper is also equipped with a gate-count operation, which enables performing automated LRE of the oracle implementations.

Our approach is briefly outlined as follows. Oracles A, b and R were provided to us in the IARPA QCS program GFI in terms of MATLAB functions, which return matrix and vector elements defining the original linear-system problem. The task was to implement them as unitary quantum circuits. We used an approach that combines "*Template Haskell*" and the "*classical-to-reversible*" functionality of Quipper, which are explained below. This approach offers a general and automated mechanism for converting classical Haskell functions into their corresponding reversible unitary quantum gates by automatically generating their inverse functions and using them to uncompute ancilla qubits.

This Section starts with a short elementary introduction to Quipper. We then proceed with demonstrating how Quipper allows automated quantum-circuit generation and manipulation and indeed offers a universal automated LRE tool. We finally discuss how Quipper's powerful capabilities have been exploited for the purpose of this work, namely achieving automated LRE of the oracles' circuit implementations.

4.1 Quipper and the circuit model

The programming language *Quipper* [14,15] is a domain-specific, higher-order, functional language for quantum computation. A snippet of Quipper code is essentially the formal description of a circuit construction. Being higher-order, it permits the manipulation of circuits as first-class citizens. Quipper is embedded in the host-language Haskell and builds upon the work of [31–35].

In Quipper, a circuit is given as a typed procedure with an input type and an output type. For example, the Hadamard and the NOT gates are typed with

```
hadamard :: Qubit -> Circ Qubit
qNOT :: Qubit -> Circ Qubit
```

They input a qubit and output a qubit. The keyword `Circ` is of importance: it says that when executed, the function will construct a circuit (in this case, a trivial circuit with only one gate).

Quantum data-types in Quipper are recursively generated: `Qubit` is the type of quantum bits; `(A,B)` is a pair of an element of type `A` and an element of type `B`; `(A,B,C)` is a 3-tuple; `()` is the unit-type: the type of the empty tuple; `[A]` is a list of elements of type `A`.

If a program has multiple inputs, we can either place them in a tuple or use the *curry notation* (→). For instance, the program

```
prog :: (A,B,C) -> Circ D
```

takes three inputs of type `A`, `B` and `C` and outputs a result of type `D`, while at the same time producing a circuit. Using the curry notation, the same program can also be written as

```
prog :: A -> B -> C -> Circ D
```

where `D` is the type of the output. We use the program by placing the inputs on the right, in order:

```
prog a b c
```

The meaning is the following: `prog a` is a function of type `B -> C -> Circ D`, waiting for the rest of the arguments; `prog a b` is a function of type `C -> Circ D`, waiting for the last argument; finally, `prog a b c` is the fully applied program. If a program has no input, it has simply the type `Circ B` if `B` is the type of its output.

Using the introduced notation, we can type the controlled-NOT gate:

```
controlled_NOT ::
Qubit -> Qubit -> Circ (Qubit,Qubit)
```

and initialization and measure:

```
qinit :: Bool -> Circ Qubit
measure :: Qubit -> Circ Bit
```

To illustrate explicitly how quantum circuits are generated with Quipper, let us use a well-known example: the EPR-pair generation, defined by the transformation $|0\rangle \otimes |0\rangle \rightarrow 1/\sqrt{2}\,(|0\rangle \otimes |0\rangle + |1\rangle \otimes |1\rangle)$. The Quipper code which creates such an EPR pair can be written as follows:

```
1   epr :: Circ (Qubit,Qubit)
2   epr = do
3      q1 <- qinit False
4      q2 <- qinit False
5      q2 <- hadamard q2
6      controlled_NOT q1 q2
7      return (q1,q2)
```

The generated circuit is presented in Fig. 20, and each line is shown with its corresponding action. Line 1 defines the type of the piece of code: `Circ` means that the program generates a circuit, and `(Qubit,Qubit)` indicates that two quantum bits are going to be returned. Line 2 starts the actual coding of the program. Lines 3 to 6 are the instructions generating new quantum bits and performing gate operations on

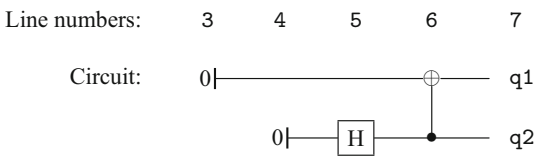

Fig. 20 EPR-pair creation; circuit generated with Quipper

them, while Line 7 states that the newly created quantum bits `q1` and `q2` are returned to the user.

Quipper is a higher-order language, that is, functions can be inputs and outputs of other functions. This allows one to build quantum-specific circuit-manipulation operators. For example,

```
controlled: (Circ A) -> Qubit -> Circ A
```

inputs a circuit, a qubit, and output the same circuit controlled with the qubit. It fails at run-time if some noncontrollable gates were used. So the following two lines are equivalent:

```
controlled (qNOT x) y
controlled_NOT x y
```

The function `classical_to_reversible`, presented in Section 4.4, is another example of high-level operator.

The last feature of Quipper useful for automated generation of oracles is the subroutine (or box) feature. The operator `box` allows macros at the circuit level: it allows re-use of the same piece of code several times in the same circuit, without having to write down the list of gates each time. When a particular piece of circuit is used several times, it makes the representation of the circuit in the memory more compact, therefore more manageable, in particular for resource estimation.

4.2 Quipper-generated resource estimation

The previous section showed how a program in Quipper is essentially a description of a circuit. The execution of a given program will generate a circuit, and performing logical resource estimation is simply achieved by completing the program with a gate-count operation at the end of the circuit-generation process. Instead of, say, sending the gates to a quantum co-processor, the program merely counts them out. Quipper comes equipped with this functionality.

4.3 Regular versus reversible computation

An oracle in quantum computation is a description of a classical structure on which the algorithm acts: a graph, a matrix, etc. An oracle is then usually presented in the form of a regular, classical function f from n to m bits encoding the problem. It is left to the reader to make this function into the unitary of Fig. 21 acting on quantum bits.

Provided that the function f is given as a procedure and not as a mere truth table, there is a known efficient strategy to build U_f out of the description of f [36].

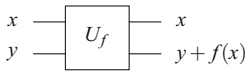

Fig. 21 General form of the oracle for a function f

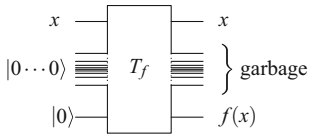

Fig. 22 Circuit T_f. Note that the middle set of inputs are ancilla qubits

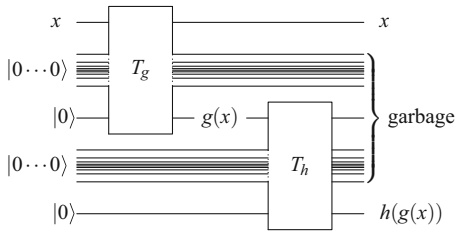

Fig. 23 Composing two oracles

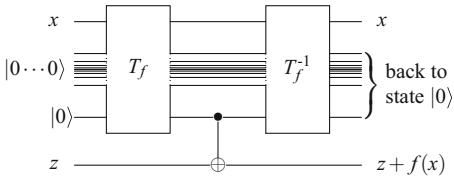

Fig. 24 Making an oracle reversible

The strategy consists in two steps. First, construct the circuit T_f of Fig. 22. Such a circuit can be built in a compositional manner as follows. Suppose that f is given in term of g and h: $f(x) = h(g(x))$. Then, provided that T_g and T_h are already built, T_f is the circuit in Fig. 23. NOT and AND are enough to write any Boolean function f: these are the base cases of the construction. The gate T_{NOT} is the controlled-NOT, and the gate T_{AND} is the Toffoli gate.

Once the circuit T_f is built, the circuit U_f, shown in Fig. 24 is simply the composition of T_f, a fanout, followed with the inverse of T_f. At the end of the computation, all the ancillas are back to 0: they are not entangled anymore and can be discarded without jeopardizing the overall unitarity of U_f.

4.4 Quipper and template Haskell

As the transformation sending a procedure f to a circuit T_f is compositional, it can be automated. We are using a feature of the host-language Haskell to perform this transformation automatically: Template Haskell. In a nutshell, it allows one to manipulate a piece of code *within the language*, produce a new piece of code and inject it in the program code. Another (slightly misleading) way of saying it is that it is a type-safe method for macros. Regardless, it allows one to do exactly what we showed in the previous section: function composition is transformed into circuit composition,

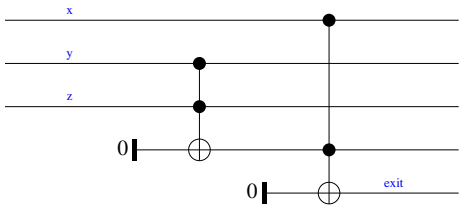

Fig. 25 Circuit mechanically generated

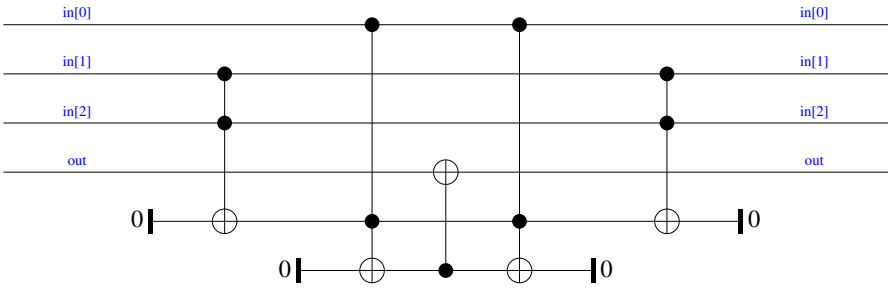

Fig. 26 Circuit made reversible

and every subfunction `f` : $A \to B$ is replaced with its corresponding circuit, whose type[14] is $A \to$ `Circ` B: a function that inputs an object of type A, builds a (piece of) circuit, and outputs B. For example, the code

```
my_and :: (Bool,Bool,Bool) -> Bool
my_and (x,y,z) = x && (y && z)
```

computing the conjunction of the three input variables x, y and z is turned into a function

```
template_my_and ::
(Qubit,Qubit,Qubit) -> Circ Qubit
```

computing the circuit in Fig. 25. Notice how the input wires are not touched and how the result is just one among many output wires. One can as easily encode the addition using binary integer.

As Quipper is a high-level language, it flawlessly allows *circuit manipulation*. In particular, one can perform the meta-operation `classical_to_reversible` sending the circuit T_f to U_f, of type

$$(A \to \texttt{Circ}\ B) \to (A, B) \to \texttt{Circ}\ (A, B),$$

provided that A and B are essentially lists of qubits, and that T_f only consists of *classical reversible* gates: NOTs, c-NOTs, cc-NOTs, etc.

In the case of our `my_and` function, it produces the circuit in Fig. 26 of the correct shape. One can easily check that the wire `out` is correctly set.

[14] Technically, the type is `Circ`$(A \to$ `Circ` $B)$. But this is only an artifact of the mechanical encoding.

```
calcRweights y nx ny lx ly k theta phi =
     let (xc',yc') = edgetoxy y nx ny in
     let xc = (xc'-1.0)*lx - ((fromIntegral nx)-1.0)*lx/2.0 in
     let yc = (yc'-1.0)*ly - ((fromIntegral ny)-1.0)*ly/2.0 in
     let (xg,yg) = itoxy y nx ny in
     if (xg == nx) then
         let i = (mkPolar ly (k*xc*(cos phi)))*(mkPolar 1.0 (k*yc*(sin phi)))*
                 ((sinc (k*ly*(sin phi)/2.0)) :+ 0.0) in
         let r = ( cos(phi) :+ k*lx )*((cos (theta - phi))/lx :+ 0.0) in i * r
     else if (xg==2*nx-1) then
         let i = (mkPolar ly (k*xc*cos(phi)))*(mkPolar 1.0 (k*yc*sin(phi)))*
                 ((sinc (k*ly*sin(phi)/2.0)) :+ 0.0) in
         let r = ( cos(phi) :+ (- k*lx))*((cos (theta - phi))/lx :+ 0.0) in i * r
     else if ( (yg==1) && (xg<nx) ) then
         let i = (mkPolar lx (k*yc*sin(phi)))*(mkPolar 1.0 (k*xc*cos(phi)))*
                 ((sinc (k*lx*(cos phi)/2.0)) :+ 0.0) in
         let r = ( (- sin phi) :+ k*ly )*((cos(theta - phi))/ly :+ 0.0) in i * r
     else if ( (yg==ny) && (xg<nx) ) then
         let i = (mkPolar lx (k*yc*sin(phi)))*(mkPolar 1.0 (k*xc*cos(phi)))*
                 ((sinc (k*lx*(cos phi)/2.0)) :+ 0.0) in
         let r = ( (- sin phi) :+ (- k*ly) )*((cos(theta - phi)/ly) :+ 0.0) in i * r
     else 0.0 :+ 0.0
```

Fig. 27 Small piece of oracle R code

4.5 Encoding oracles

The oracles of QLSA were given to us as a set of MATLAB functions as part of the IARPA QCS program GFI. These functions computed the matrix A and the vectors b and R of [5]. They were not using any particular library: directly translating them into Haskell was a straightforward operation. As the MATLAB code came with a few tests to validate the implementation, by running them in Haskell we were able to validate our translation.

The main difficulty was not to translate the MATLAB code into Quipper, but rather to encode by hand the real arithmetic and analytic functions that were used. Figure 27 shows a snippet of translated Haskell code: it is a nontrivial operation using trigonometric functions. Another part of the oracle is also using arctan.

To be able to be processed through Template Haskell, all the arithmetic and analytic operations had to be written from scratch on integers encoded as lists of `Bool`. We used an encoding on fixed-point arithmetic. Integers were coded as 32-bit plus one bit for the sign, and real numbers as 32-bit integer part and 32-bit mantissa, plus one bit for the sign. We could have chosen to use floating-point arithmetic, but the operations would have been much more involved: the corresponding generated circuit would have been even bigger.

We made heavy use of the subroutine facility of Quipper: All of the major operations are boxed, that is, appear only once in the internal structure representing the circuit. This allows manageable processing (e.g., printing, or resource counting). As an example, the circuit for Oracle R of QLSA is shown in Fig. 28.

4.6 Compactness of the generated oracles

Our strategy for generating circuits with Template Haskell is *efficient* in the following sense: the size of the generated quantum circuit is exactly the same as the *number of*

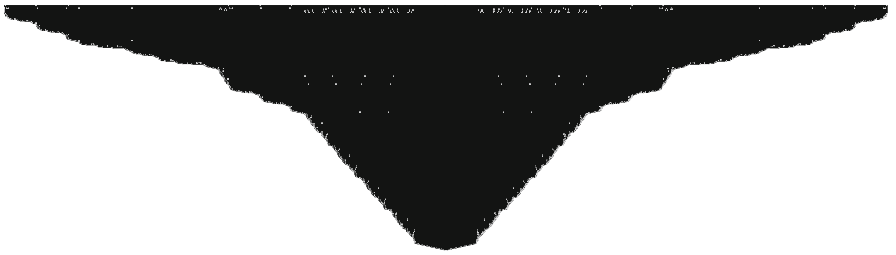

Fig. 28 Oracle R, automatically generated. In the on-line version of the paper the reader can magnify the PDF image to see the details of the circuit. For display purposes, in this figure we use one wire for integers and two wires for real numbers. Only the main structure is shown: all operations such as tests, arithmetic, and analytic operations only appear as named boxes

steps in the classical program. For example, if the classical computation consists of n conjunctions and m negations, the generated quantum circuit consists of n Toffoli gates and m CNOT gates.

The advantage of this technique is that it is fully general: with this procedure, any classical computation can be turned into an oracle in an efficient manner.

Optimizing oracle sizes As we show in this paper, the sizes of the generated oracles are quite impressive. In the current state of our investigations, we believe that, even with hand-coding, these numbers could only be improved upon by a factor of 5, or perhaps at most a factor of 10. We think that accomplishing a greater reduction beyond these moderate factors would require a drastic change in the generation approach and techniques.

The reason why we think it is possible to achieve the mentioned moderate optimization is the following. Although the oracles we deal with in this work are specified and tailored to the particular problem we have been analyzing, they are also general in the sense that they are made of smaller algorithms (e.g., adders, multipliers …). The reversible versions of these algorithms have been studied for a long time, and quite efficient proposals have been made. An analysis of the involved resources shows that for the addition of n-bit integers, the number of gates involved in the automatically generated adder gate T_f is $\lesssim 25n$ and the number of ancillas is $\lesssim 8n$. A *hand-made* reversible adder can be constructed [37] with, respectively, $\lesssim 5n$ gates and $\leq n$ ancillas. If one found a way to reuse these circuits in place of our automatically generated adders, it would reduce the oracle sizes. However, it could only do so by a relatively small factor; the total number of gates would still be daunting.

Despite this drawback, our method is versatile and able to provide circuits for any desired function f without further elaborate analysis.

5 Results

Our LRE for QLSA for problem size $N = 332{,}020{,}680$ is summarized in Table 2. The following comments explain this table and our assumptions.

Table 2 Resource requirements of QLSA for the problem size $N = 332{,}020{,}680$ and algorithmic accuracy $\varepsilon = 0.01$

Resources	Incl. oracles	Excl. oracles
Max. overall number of qubits in use at a time	3×10^8	341
Max. number of data qubits at a time	60	60
Max. number of ancilla qubits in use at a time	3×10^8	281
Overall number of ancilla generation-use-termination cycles	2.8×10^{27}	8.2×10^{21}
Total number of gates	2.37×10^{29}	3.34×10^{25}
# H gates	2.7×10^{28}	1.20×10^{25}
# S gates	1.4×10^{28}	6.3×10^{24}
# T gates	9.5×10^{28}	1.29×10^{25}
# X gates	1.6×10^{28}	2.0×10^{23}
# Z gates	2.4×10^{23}	2.4×10^{23}
# CNOT gates	8.5×10^{28}	1.7×10^{24}
Circuit width	3×10^8	341
Circuit depth	1.8×10^{29}	3.30×10^{25}
T-Depth	8.2×10^{28}	1.28×10^{25}
Measurements	2.8×10^{27}	8.23×10^{21}

Unlike with QEC protocols where the distinction between "data qubits" and "ancilla qubits" is clear, here this distinction is somewhat ambiguous; indeed, all qubits involved in the algorithm are initially prepared in state $|0\rangle$, and some qubits that we called ancilla qubits exist from the start to the end of a full quantum-computation part (such as e.g., single-qubit registers R_6, R_8). We regard qubits which carry the data of the linear-system problem and store its solution at the end of the quantum computation as data qubits; they constitute the quantum data registers R_2 and R_3, see Table 1. All other qubits, including those of QAE and HS control registers R_0 and R_1 as well as of the computational registers R_4 and R_5, are considered ancilla qubits.

It is important to note that the overall QLS algorithm consists of *four* independent quantum-computation parts, namely the four calls of "*AmpEst*" subalgorithms, see Fig. 2, while the top-level function "*QLSA_main*" performs a classical calculation of the RCS (by Eq. (16)) using the results $\phi_b, \phi_x, \phi_{r0}, \phi_{r1}$ of its four quantum-computation parts. These four independent "AmpEst" subalgorithms can either be performed in parallel or sequentially, and the actual choice should be subject to any time/space trade-off considerations. Here we assume a sequential implementation, so that data and ancilla qubits can be reused by the four amplitude estimation parts. Hence, the qubit counts provided in Table 2 represent the maximum number of qubits in use at a time required by the most demanding of the four independent "AmpEst" subalgorithms. The maximum overall number of qubits (data and ancilla) in use at a time is also the definition for *circuit width*. While with a sequential implementation we aim at minimizing the circuit width (space consumption), we can do so only at the cost of increasing the circuit depth (time consumption). The overall circuit depth is the sum

of the depths of the four "AmpEst" subalgorithms. By a brief look at Fig. 2 it is clear that the circuit depths are similarly large for "AmpEst_ϕ_x" and "AmpEst_ϕ_r" (where the latter is called twice), whereas compared to these the circuit depth of "AmpEst_ϕ_b" is negligible. Hence the overall circuit depth is roughly three times the circuit depth of subalgorithm "AmpEst_ϕ_r." We could just as well assume a parallel implementation of the four "AmpEst" calls. In this case the overall circuit depth would be by a factor $1/3$ smaller than in the former case. However, this circuit depth decrease can only be achieved at the cost of incurring a circuit-width increase. We would need up to four copies of the quantum registers listed in Table 1, and the required number of data and ancilla qubits in use at a time would be larger by a factor that is somewhat smaller than four.

QLSA has numerous iterative operations (in particular due to Suzuki-Higher-Order Integrator method with Trotterization) involving ancilla-qubit "*generation-use-termination*" *cycles*, which are repeated, over and over again, while computation is performed on the same end-to-end data qubits. Table 2 provides an estimate for both the number of ancilla qubits employed at a time and for the overall number of ancilla generation-use-termination cycles executed during the implementation of all the four "AmpEst" subalgorithms. To illustrate the difference we note that, for some quantum-computer realizations, the physical information carriers (carrying the ancilla qubits) can be reused, for others however, such as photon-based quantum-computer realizations, the information carriers are lost and have to be created anew.

Furthermore, the gate counts actually mean the number of elementary logical *gate operations*, independent of whether these operations are performed using the same physical resources (lasers, interaction region, etc.) or not. The huge number of measurements results from the vast overall number of ancilla-qubit uses; after each use an ancilla has to be uncomputed and eventually terminated to ensure reversibility of the circuit. Finally, Table 2 distinguishes between the overall LRE that includes the oracle implementation and the LRE for the bare algorithm with oracle calls regarded as "for free" (excluding their resource requirements).

6 Discussion

6.1 Understanding the resource demands

Our LRE results shown in Table 2 suggest that the resource requirements of QLSA are to a large extent dominated by the quantum-circuit implementation of the numerous oracle A queries and their associated resource demands. Indeed, accounting for oracle implementation costs yields resource counts which are by several orders of magnitude larger than those if oracle costs are excluded. While Oracle A queries have only slightly lower implementation costs than Oracle b and Oracle R queries, it is the number of queries that makes a substantial difference. As clearly illustrated in Fig. 2, Oracle A (required to implement the Hamiltonian transformation e^{iAt} with $t \leq t_0 \sim O(\kappa/\varepsilon)$) is queried by many orders of magnitude more frequently than Oracles b and R, which are needed only for preparation of the quantum states $|b\rangle$ and $|R\rangle$ corresponding to the column vectors $\mathbf{b}, \mathbf{R} \in \mathbb{C}^N$. Hence, the overall LRE of the algorithm depends

very strongly on the Oracle A implementation. However, note that Oracles b and R contribute most to circuit width due to the vast number of ancilla qubits ($\sim 3 \times 10^8$) they employ at a time, see Table 10 in "Appendix 3."

The LRE for the bare algorithm, i.e., with oracle queries and "IntegerInverse" function regarded as "for free" (excluding their resource costs), amounts to the order of magnitude 10^{25} for gate count and circuit depth—still a surprisingly high number. In what follows, we explain how these large numbers arise, expanding on all the factors in more detail that yield a significant contribution to resource demands. To do so, we make use of Fig. 2.

QLSA's LRE is dominated by series of nested loops consisting of numerous iterative operations, see Fig. 2. The major iteration of circuits with similar resource demands occurs due to the Suzuki-Higher-Order Integrator method including a Trotterization with a large time-splitting factor of order 10^{12} to accurately implement each run of the HS as part of QPEA. Indeed, each single call of "HamiltonianSimulation" yields the iteration factor $r = 2.5 \times 10^{12}$. This subroutine is called *twice* during the "Solve_x" procedure, and the latter is furthermore employed *twice* within the (controlled) Grover iterators in three of the *four* QAEAs. There are $\sum_{j=0}^{n_0-1} 2^j = 2^{n_0} - 1 = 16{,}383$ controlled Grover iterators employed within each of the four QAEAs. Hence, the "HamiltonianSimulation" subroutine is employed $(2^{n_0} - 1) \times 4 \times 3 = 196{,}596 \approx 2 \times 10^5$ number of times altogether. Because each of its calls uses Trotterization with time-splitting factor 2.5×10^{12} and a Suzuki-Higher-Order Integrator decomposition with order $k = 2$ involving a further additional factor 5, we already get the factor $\sim 2.5 \times 10^{18}$. Moreover, the lowest-order Suzuki operator is a product of $2 \times N_b = 18$ one-sparse Hamiltonian propagator terms (where $N_b = 9$ is the number of bands in matrix A); each such term calls the "HsimKernel" function, with "band" and "timestep" as its runtime parameters. In addition, each call of HsimKernel employs Oracle A *six* times and furthermore involves 24 applications of the procedure "Hmag" controlled by the time register R_1. Thus, in total QLSA involves $6 \times 18 \times 2.5 \times 10^{18} \approx 2.7 \times 10^{20}$ Oracle A queries and $24 \times 18 \times 2.5 \times 10^{18} \approx 10^{21}$ calls of controlled Hmag. Hence, even if subroutine Hmag consisted of a single gate and oracle A queries were for free, we would already have approx. 10^{21} for gate count and circuit depth.

However, Hmag is a subalgorithm consisting of further subcircuits to implement the application of the magnitude component of a particular one-sparse Hamiltonian term to an arbitrary state. It consists of several W gates, Toffolis and controlled rotations. Hence, a further increase of the order of magnitude is incurred by various decompositions of multicontrolled gates and/or rotation gates into the elementary set of fault-tolerant gates $\{H, S, T, X, Z, \text{CNOT}\}$, using the well-known decomposition rules outlined in "Appendix 2" (e.g., optimal-depth decompositions for Toffoli [38] and for controlled single-qubit rotations [39–42]). In our analysis, this yields a further factor $\sim 10^4$. Thus, even if we exclude oracle costs, we have $10^{21} \times 10^4 = 10^{25}$ for gate count and circuit depth for the bare algorithm, simply because of a large number of iterative processes (due to Trotterization and Grover-iterate-based QAE) combined with decompositions of higher-level circuits (such as multicontrolled NOTs) into elementary gates and single-qubit rotation decompositions (factors $\sim 10^2$–10^4).

If we include the oracle implementation costs, the dominant contribution to LRE is that of Oracle A calls, because oracle A is queried by a factor $\sim 10^{15}$ more frequently than Oracle b and even by a larger factor than Oracle R. Each Oracle A query's circuit implementation has a gate count and circuit depth of order $\sim 2.5 \times 10^8$, see "Appendix 3." Having approx. 2.7×10^{20} Oracle A queries, the LRE thus amounts to the order of magnitude $\sim 10^{29}$.

Let us briefly summarize the nested loops of QLSA that dominate the resource demands, while other computational components have negligible contributions. The dominant contributions result from those series of nested loops which include Hamiltonian Simulation as the most resource-demanding bottleneck. The outer loops in these series are the first-level QAEA subroutines to find estimates for ϕ_x, ϕ_{r0} and ϕ_{r1}, each involving $2^{n_0} - 1 = 16{,}383$ controlled Grover iterators. Each Grover iterator involves several implementations of Hamiltonian Simulation based on Suzuki-Higher-Order Integrator decomposition and Trotterization with $r \approx 10^{12}$ time-splitting slices. Each Trotter slice involves iterating over each matrix band whereby the corresponding part of Hamiltonian evolution is applied to the input state. Finally, for each band several oracle A implementations are required to compute the corresponding matrix elements, which moreover employs several arithmetic operations, each of which themselves require loops with computational effort scaling polynomially with the number of bits in precision.

6.2 Comparison with previous "big-O" estimations

As pointed out in the Introduction, we provide the first *concrete* resource estimation for QLSA in contrast to the previous analyses [3,5] which estimated the run-time of QLSA only in terms of its asymptotic behavior using the "big-O" characterization. As the latter is supposed to give some hints on how the size of the circuit evolves with growing parameters, it is interesting to compare our concrete results for gate count and circuit depth with what one would expect according to the rough estimate suggested by the big-O (complexity) analysis. The big-O estimations proposed by Harrow et al. [3] and Clader et al. [5] have been briefly discussed in the Introduction and are given in Eqs. (1) and (3), respectively.

Complexity-wise, the parameters taken into account in the big-O estimations are the size N of the square matrix A, the condition number κ of A, the sparseness d which is the number of nonzero entries per row/column in A, and the desired algorithmic accuracy given as error bound ε. The choice of parameters made in this paper fixes these values to $N = 332{,}020{,}680$, $\kappa = 10^4$, $d = 7$, and $\varepsilon = 10^{-2}$. If one plugs them into Eqs. (1) and (3), one gets, respectively, $\sim 4 \times 10^{12}$ and $\sim 2 \times 10^{12}$.

Although these numbers are large, they are not even close to compare with our estimates. This is due to the way a big-O estimate is constructed: it only focuses on a certain set of parameters, the other ones being roughly independent of the chosen set. Indeed, the "function" provided as big-O estimate is only giving a trend on how the estimated quantity behaves as the chosen set of parameters goes to infinity (or to zero, in the case of ε). Hence, only the limiting behavior of the estimate can be predicted with high accuracy, when the chosen relevant parameters it depends on tend toward

particular values or infinity, while the estimate is very rough for other values of these parameter. In particular, a big-O estimate is hiding a set of constant factors, which are unknown. In the case of QLSA, our LRE analysis does not reveal a trend, it only gives one point. Nonetheless, it shows that these factors are extremely large, and that they must be carefully analyzed and otherwise taken into account for any potentially practical use of the algorithm.

Although the (unknown) constant factors implied by big-O complexity cannot be inferred from our LRE results obtained for just a single problem size, we can nevertheless consider which steps in the algorithm are likely to contribute most to these factors. With our fine-grained approach we found that, if excluding the oracle A resources, the accrued circuit depth $\sim 10^{25}$ is roughly equal to $3 \times (2^{n_0} - 1)$ Grover iterations (as part of amplitude estimation loops for ϕ_x, ϕ_{r0} and ϕ_{r1}) *times* $4 \times (2N_b) \times 5 \times 2.5 \times 10^{12}$ for the number of exponentials needed to implement the Suzuki-Trotter expansion (as part of implementing HS, which is employed twice in Solve_x that is again employed twice in each Grover iterator) *times* a factor $\sim 24 \times 10^4$ coming about from the circuits to implement, for each particular A_j in the decomposition [Eq. (8)], the corresponding part of Hamiltonian state transformation. In terms of CJS big-O complexity the circuit depth is $\widetilde{O}\left(\kappa d^7 \log(N)/\varepsilon^2\right)$, which comes from $\widetilde{O}\left(1/\varepsilon\right)$ QAE Grover iterations,[15] *times* $\widetilde{O}\left(d^4\kappa/\varepsilon\right)$ exponential operator applications to implement the Suzuki-Trotter expansion,[16] *times* $O\left(\log N\right)$ oracle A queries to simulate each query to any A_j in the decomposition [Eq. (8)], *times* the overhead of $O(d^3)$ computational steps including $O(d^2)$ Oracle A queries to estimating the preconditioner M of the linear system in order to prepare the preconditioned state $M|b\rangle$, see [5]. Here it is appropriate to note though that the HHL and CJS runtime complexities given in Eqs. (1) and (3), respectively, neglect more slowly growing terms, as indicated by the tilde notation $\widetilde{O}(\cdot)$. However, in a comparison with our empirical gate counts we ought to also take those slowly growing terms into account. For instance, there is another factor of $(\kappa d^2/\varepsilon^2)^{1/4} \approx 3 \times 10^2$ contributing to the number of Suzuki-Trotter expansion slices, which was ignored in the $\widetilde{O}$ notation for HHL and CJS complexities, while it was accounted for in our LRE. By inspecting and comparing (CJS big-O vs. our LRE) the orders of magnitude of the various contributing terms, we conclude that the big-O complexity is roughly two orders of magnitude off (smaller) from our empirical counts for the Suzuki-Trotter expansion step. As for the QAE steps, our LRE count is $\sim 5 \times 10^4$, which is roughly two orders of magnitude higher than $O(1/\varepsilon)$ and smaller than $O(\kappa/\varepsilon)$, suggesting that $O(1/\varepsilon)$ is too optimistic while $O(\kappa/\varepsilon)$ is too conservative. Finally, the big-O complexity misses roughly 5 orders of magnitude that our fine-grained approach reveals for the circuit implementation of the Hamiltonian state transformation for each A_j at the lowest algorithmic level.

[15] However, see our remarks in footnotes 6 and 11 in which we pointed out that $O(\kappa/\varepsilon)$ may be a more appropriate estimate for the complexity of the QAE loops.

[16] For a d-sparse A, simulating $\exp(iAt)$ with additive error ε using HS techniques [8] requires a runtime proportional to $d^4 t (t/\varepsilon)^{o(1)} \equiv \widetilde{O}\left(d^4 t\right)$, see [3,8]. It is performing the *phase estimation* (as part of "Solve_x"), which is the dominant source of error, that requires to take $t_0 = O(\kappa/\varepsilon)$ for the various times $t = \tau t_0/T$ defining the HS control register in order to achieve a final error smaller than ε, see [3].

In order to understand what caused such large constant factors, we estimated the resources needed to run QLSA for a smaller problem size[17] while keeping the same precision (and therefore the same size for the registers holding the computed values). Specifically, we chose $N = 24$, while we kept the condition number and the error bound at the same values $\kappa = 10^4$ and $\varepsilon = 10^{-2}$, respectively. Despite the fact that the matrix A lost several orders of magnitude in size, the circuit width and depth ended up being of roughly the same order of magnitude as of Table 2.

What our results suggest is that the large constant factors arise as a consequence of the desired precision forcing us into choosing large sizes for the registers, whereas the LRE is not notably impacted by a change in problem size N. This can intuitively be understood as follows. First, the total number of gates required for QLSA's nonoracle part scales as $O(\log N)$, cf. Eq. (3); hence, using $N = 24$ in place of $N = 332{,}020{,}680$ suggests an LRE reduction only by a moderate factor $\sim$5. Secondly, what matters for the LRE of oracles is also mostly determined by the desired accuracy ε. Each oracle query essentially computes a single (complex) value corresponding to a particular input from the set of all inputs. The oracles are oblivious to the problem size and to the actual value of each of their inputs. While oracles obtain actual input data from the data register R_2 or R_3, whose size $n_2 = n_3 = \log_2(2N)$ clearly depends on N, these are not the ones that crucially determine the oracles' sizes. What virtually matters for the size of the generated quantum circuit implementing an oracle query, is the size of the *computational registers* R_4 and R_5 used to compute and hold the output value of each particular oracle query. In our analysis, these registers have size $n_4 = 65$, cf. Table 1; they were kept at the same size when computing QLSA's LRE for the smaller problem size $N = 24$.

6.3 Lack of parallelism

Comparing the estimates for the total number of gates and circuit depth reveals a distinct *lack of parallelism*[18] in the design of QLSA. As explained earlier, due to the

[17] A smaller problem size is obtained by reducing the spatial domain size of the electromagnetic scattering FEM simulation, via reductions in parameters n_x and n_y which represent the number of FEM vertices in x and y dimensions. The immediate consequence is a reduction of the common length of quantum data registers R_2 and R_3, i.e., $n_2 = \lceil \log_2(2N) \rceil$, where $N = n_x(n_y - 1) + (n_x - 1)n_y$. Such register-length reduction is expected to affect the resource requirements for all oracles as well as all subroutines that involve the data registers R_2 and R_3. In fact, the input registers to all oracles are of length n_2, and shortening them has the potential of reducing the oracle sizes. However, we recounted oracles' resources using Quipper, with $n_2 = 6$ in place of $n_2 = 30$, and found that the only difference involves the number of ancillas and measurements required. When checking the resource change of the entire QLSA circuit, we found negligible difference. Indeed, changes in n_2 have a relatively little effect on resources of the bare algorithm (excluding oracle costs), because the dominant contribution to resources in the nonoracle part is given by the time-splitting factor imposed by Hamiltonian-evolution simulation, which does not directly depend on n_2. Besides, since the total number of operations required for QLSA's nonoracle part has a complexity that scales logarithmically in N, see Eqs. (1) and (3), the resources for $n_2 = 6$ in place of $n_2 = 30$ are expected to diminish by just a relatively small factor $\sim$5.

[18] One can get a sense of the amount of parallelism of the overall circuit by comparing the total number of gates of an algorithm to its circuit depth. In our analysis, they only differ by a factor of $\sim$1.33 if oracles are included, and by a factor of $\sim$1.01 if oracles are excluded, thus most of the gates must be being applied sequentially.

highly repetitive structures of the algorithm primitives used, most of the gates have to be performed sequentially. Indeed, QLSA involves numerous iterative operations. The major iteration of circuits with similar resource requirements occurs due to the Suzuki-Higher-Order Integrator method that also involves Trotterization, which uses a large time-splitting factor of order 10^{12} to accurately implement each run of the Hamiltonian-evolution simulation. In fact, the iteration factor imposed by Trotterization of the Hamiltonian propagator is currently a hard bound on the overall circuit depth and even the total LRE of QLSA, and it crucially depends on the aimed algorithmic precision ε. The remarks in the following paragraph expand on this issue in more detail.

6.4 Hamiltonian-evolution simulation as the actual bottleneck and recent advancements

It is worth emphasizing that the quantum-circuit implementation of the Hamiltonian transformation e^{iAt} using well-established HS techniques [8] constitutes the actual bottleneck of QLSA. Indeed, this step implies the largest contribution to the overall circuit depth; it is given by the factor $r \times 5^{k-1} \times (2N_b)$, see Fig. 2, which is imposed by the Suzuki-Higher-Order Integrator method together with Trotterization. According to Eq. (17) and the discussion following it, $r \sim O\left((N_b\kappa)^{1+1/2k}/\varepsilon^{1+1/k}\right)$. Thus, the key dependence of the time-splitting factor r is on the condition number κ and the error bound ε rather than on problem size N. The dependence on the latter enters only through the number of bands N_b (in the general case, the number m of submatrices in the decomposition [Eq. (8)]), which can be small even for large matrix sizes, as is the case in our example. This feature explains why we can get similar LRE results for $N = 332{,}020{,}680$ and $N = 24$ if κ and ε are kept at the same values for both cases and the number of bands N_b is small (see above).

It is also important to note that there has been significant recent progress on improving HS techniques. Berry et al. [43] provide a method for simulating Hamiltonian evolution with complexity polynomial in $\log(1/\varepsilon)$ (with ε the allowable error). Even more recent works by Berry et al. [44,45] improve upon results in [43] providing a quantum algorithm for simulating the dynamics of sparse Hamiltonians with complexity sublogarithmic in the inverse error. Compared to [44], the analysis in [45] yields a near-linear instead of superquadratic dependence on the sparsity d. Moreover, unlike the approach [43], the query complexities derived in [44,45] are shown to be independent of the number of qubits acted on. Most importantly, all three approaches [43–45] provide an exponential improvement upon the well-established method [8] that our analysis is based on.[19] To account for these recent achievements, we estimate the impact they may have with reference to the baseline imposed by our LRE results. The modular nature of our LRE approach allows us to do this estimation. The following back-of-the-envelope evaluation shows that, for $\varepsilon = 0.01$, the advanced HS

[19] The recently published advanced HS approaches [43–45] that promise more resource-efficient computation were not available at the time when our detailed implementation of QLSA and the corresponding LRE analysis (for which we used the previously published HS techniques [8]) were performed.

approaches [43], [44] and [45] may offer a potential reduction of circuit depth and overall gate count by orders of magnitude 10^1, $\sim 10^4$ and $\sim 10^5$, respectively.

Indeed, let us compare the scalings of the total number of one-sparse Hamiltonian-evolution terms required to approximate e^{iAt} to within error bound $\varepsilon = 0.01$ for the prior approach [8] (used here) and the recent methods [43,45]. In doing so, we arrive at contrasting

$$8m5^{2k-3/2}(m\|A\|t)^{1+1/2k}/\varepsilon^{1/2k} \tag{19}$$

$$\text{vs.} \quad O\left([d^2\|A\|t + \log(1/\varepsilon)]\log^3[d\|A\|t/\varepsilon]n^c\right) \tag{20}$$

$$\text{or} \quad O\left(d\|A\|t\frac{\log(d\|A\|t/\varepsilon)}{\log\log(d\|A\|t/\varepsilon)}\right) \tag{21}$$

for the three approaches [8], [43] and [45], respectively. In the first term, m denotes the number of submatrices in the decomposition [Eq. (8)]; in the general case, $m = 6d^2$, in our toy-problem analysis, $m = N_b$. In the second and third term, d is the sparsity of A, and n is the number of qubits acted on, while c is a constant. In all three expressions, $\|A\|$ is the spectral norm of the Hamiltonian A, which in our toy-problem example is time-independent. As stated in Sect. 3.4.5, for QLSA to be accurate within error bound ε, we must have $\|A\|t \sim O(\kappa/\varepsilon)$, cf. [3]. Using $\|A\|t \leq \|A\|t_0 = 7 \times \kappa/\varepsilon$ and the parameter values $m = N_b = 9, k = 2, d = 7, n = n_2 = 30$ and $c \geq 1$, expression (19) yields $\sim 7\times 10^{13}$, whereas the query complexity estimates (20) and (21) yield $\gtrsim 5\times 10^{12}$ and $\sim 5\times 10^8$, respectively. Hence, notably the advanced results in [45] imply that an improvement of our LRE by order of magnitude $\sim 10^5$ seems feasible.

7 Conclusion

A key research topic of quantum-computer science is to understand what computational resources would actually be required to implement a given quantum algorithm on a realistic quantum computer, for the large problem sizes for which a quantum advantage would be attainable. Traditional algorithm analyses based on big-O complexity characterize algorithmic efficiency in terms of the asymptotic leading-order behavior and therefore do not provide a detailed accounting of the *concrete* resources required for any given specific problem size, which however is critical to evaluating the practicality of implementing the algorithm on a quantum computer. In this paper, we have demonstrated an approach to how such a concrete resource estimation can be performed.

We have provided a detailed estimate for the logical resource requirements of the quantum linear-system algorithm, which under certain conditions solves a linear system of equations, $A\mathbf{x} = \mathbf{b}$, exponentially faster than the best known classical method. Our estimates correspond to the explicit example problem size beyond which the quantum linear-system algorithm is expected to run faster than the best known classical linear-system solving algorithm. Our results have been obtained by a combination of manual analysis for the bare algorithm and automated resource estimates for ora-

cles generated via the quantum programming language Quipper and its compiler. Our analysis shows that for a desired calculation precision accuracy $\varepsilon = 0.01$, an approximate circuit width 340 and circuit depth of order 10^{25} are required if oracle costs are excluded, and a circuit width and circuit depth of order 10^8 and 10^{29}, respectively, if the resource requirements of oracles are taken into account, showing that the latter are substantial. We stress once again that our estimates pertain only to the resource requirements of a single run of the complete algorithm, while actually multiple runs of the algorithm are necessary (followed by sampling) to produce a reliable accurate outcome.

Our LRE results for QLSA are based on well-established quantum computation techniques and primitives [1,6–8,22] as well as our approach to implement oracles using Quipper. Hence, our estimates strongly rely on the efficiency of the applied methods and chosen approach. Improvement upon our estimates can only be achieved by advancements enabling more efficient implementations of the utilized quantum-computation primitives and/or oracles. For example, as pointed out in Sect. 6, most recent advancements of Hamiltonian-evolution simulation techniques [45] suggest that a substantial reduction of circuit depth and overall gate count by order of magnitude $\sim 10^5$ seems feasible. Likewise, more sophisticated methods to generate quantum-circuit implementations of oracles more efficiently may become available. We think though that significant improvements are going to come from inventing a better QLS algorithm, or more resource-efficient Hamiltonian-evolution simulation approaches, rather than from improvements to Quipper. While we believe that our estimates may prove to be conservative, they yet provide a well-founded "*baseline*" for research into the reduction of the algorithmic-level minimum resource requirements, showing that a reduction by many orders of magnitude is necessary for the algorithm to become practical. Our modular approach to analysis of extremely large quantum circuits reduces the cost of updating the analysis when improved quantum-computation techniques are discovered.

To give an idea of how long the algorithm would have to run at a minimum, let us suppose that, in the ideal case, all logic gates take the same amount of time τ, and have perfect performance thus eliminating the need for QC and/or QEC. Then for any assumed gate time τ, one can calculate a lower limit on the amount of time required for the overall implementation of the algorithm. For example, if $\tau = 1$ns (which is a rather optimistic assumption; for other gate duration assumptions, one can then plug in one's own assumptions), a circuit depth of order 10^{25} (10^{29}) would correspond to a run-time approx. 3×10^8 (3×10^{12}) years, which apparently compares with or even exceeds the age of the Universe (estimated to be approx. 13.8×10^9 years). Even with the mentioned promising improvements by a factor $\sim 10^5$ for the Hamiltonian-evolution simulation and by a factor ~ 10 for the oracle implementations, we would still deal with run-times approx. 3×10^2 (3×10^6) years.

Although our results are surprising when compared to a naive analysis of the previous big-O estimations of the algorithm [3,5], the difference can be explained by the factors hidden in the big-O estimation analyses: we infer that these factors come for the most part from the large register sizes, chosen because of the desired precision.

The moral of this analysis is that quantum algorithms are not typically designed with implementation in mind. Considering only the overall coarse complexity of a given

algorithm does not make it automatically feasible. In particular, our analysis shows that book-keeping parameters such as the size of registers have to be considered.

Our analysis highlights an avenue for future research: quantum programming languages and formal methods. In computer science, mature techniques have been developed for decades, and we ought to adapt and implement them for a fine-grained analysis of quantum algorithms to pinpoint the various parameters in play and their relationships. In particular, these techniques may also allow to explicitly identify the actual bottlenecks of a particular implementation and provide useful insights on what to focus on for optimizations: in the case of QLSA, for instance, the Hamiltonian-evolution simulation and oracle implementations. Combining a fine-grained approach with asymptotic big-O analysis, a much fuller understanding of the bottlenecks in quantum algorithms emerges enabling focused research on improved algorithmic techniques.

Acknowledgements This work was accomplished as part of the PLATO Project: "Protocols, Languages and Tools for resource-efficient Quantum Computation," which was conducted within the scope of IARPA Quantum Computer Science (QCS) program and derived some of its goals from that source. PLATO was performed jointly by Applied Communication Sciences (ACS), Dalhousie University, the University of Pennsylvania, Louisiana State University, Southern Illinois University, and the University of Massachusetts at Boston. We would like to thank all PLATO team members for insightful discussions. Supported by the Intelligence Advanced Research Projects Activity (IARPA) via Department of Interior National Business Center Contract Number D12PC00527. The US Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon.

Appendix 1: Single-qubit unitaries in terms of pre-specified elementary gates

Implementation according to work by A. Fowler

To convert *any* single-qubit unitary to a circuit in terms of a pre-specified set of gates $\{X, Y, Z, H, S, T\}$, we could use the famous *Solovay–Kitaev* algorithm, see, e.g., [1] and references therein. However, this work can result in unnecessarily long global phase correct approximating sequences, since the trace-norm used in the Solovay–Kitaev theorem does not ignore global phases. Some optimizations of the Solovay–Kitaev algorithm are possible, see e.g., [46]. For the single-qubit rotation gates, we base our estimates on work by A. Fowler (see [39], p. 125 and [40]). This work constructs optimal fault-tolerant approximations of single-qubit phase rotation gates

$$R_{\pi/2^d} := \begin{pmatrix} 1 & 0 \\ 0 & e^{i\pi/2^d} \end{pmatrix}. \tag{22}$$

Fowler shows that a phase rotation by an angle of $\pi/128$ can be approximated by a sequence of fault-tolerant gates with a distance measure

$$\mathrm{dist}(R_{\pi/128}, U_{46}) := \sqrt{\frac{m - \left|\mathrm{Tr}(R_{\pi/128}^{\dagger} U_{46})\right|}{m}} \approx 7.5 \times 10^{-4} < 0.01 \tag{23}$$

by choosing U_{46} as follows:

$$\begin{aligned} U_{46} = {} & HTHTHT(SH)THT(SH)T(SH)T(SH)THT \\ & (SH)T(SH)THTHT(SH)T(SH)THT(SH)T \\ & (SH)T(SH)THT(SH)THT(HS^{\dagger})T \end{aligned} \tag{24}$$

This sequence contains 23 H gates, 23 T ($\pi/8$) gates and 13 S or $S^{\dagger}$ gates. In general, the approximating sequence is of the form $G_i T G_j T \ldots$, where $G_i, G_j \in \mathscr{G}$, a precomputed set of gates, which together with the Identity gate I form a group under multiplication $\{I, G_1, G_2, \ldots, G_{23}\}$. Here, $G_1 = H, G_2 = X, G_3 = Z, G_4 = S, G_5 = S^{\dagger}, G_6 = XH, G_7 = ZH, G_8 = SH, G_9 = S^{\dagger}H, G_{10} = ZX, G_{11} = SX, G_{12} = S^{\dagger}X, G_{13} = HS, G_{14} = HS^{\dagger}, G_{15} = ZXH, G_{16} = SXH, G_{17} = S^{\dagger}XH, G_{18} = HSH, G_{19} = HS^{\dagger}H, G_{20} = HSX, G_{21} = HS^{\dagger}X, G_{22} = S^{\dagger}HS, G_{23} = SHS^{\dagger}$. To represent the complete set of approximating sequences, Fowler includes $G_{24} = T$.

The sequence given in Eq. (24) contains 46 G_j gates. The number of T gates is 23, or half the length of the approximating sequence in terms of G_j gates. The number of H gates in this particular sequence is also 23, and the rest of the 59 elementary gates are S (or $S^{\dagger}$) gates.

Fowler also investigated the approximation of arbitrary single-qubit gates

$$U = \begin{pmatrix} \cos(\theta/2)e^{i(\alpha+\beta)/2} & \sin(\theta/2)e^{i(\alpha-\beta)/2} \\ -\sin(\theta/2)e^{i(-\alpha+\beta)/2} & \cos(\theta/2)e^{i(\alpha+\beta)/2} \end{pmatrix} \tag{25}$$

by sequences of gates from the group $\mathscr{G}$. 1000 random matrices were chosen, with α, β and θ chosen uniformly in $[0, 2\pi)$. Optimal approximations U_l were constructed for each random matrix, and a line was fitted to the average distance $\mathrm{dist}(U, U_l)$ plotted for each l. Fowler obtained the following fit for the average number l of single-qubit fault-tolerant gates required to obtain a fault-tolerant approximation of an arbitrary single-qubit unitary to within the distance:

$$\delta = \mathrm{dist}(U, U_l) = 0.292 \times 10^{-0.0511 \cdot l}. \tag{26}$$

In other words, to obtain a distance δ *on average*, we need *on average* $l = \frac{\log_{10}(\delta/0.292)}{-0.0511}$ gates. For $\delta = 7.5 \times 10^{-4}$, we obtain $l = 50.69$. Compare this to the exact result $l = 46$ for $R_{\pi/128}$. Also, we note that 46 G_j gates correspond to 59 elementary gates, of which 23 are T gates. For 51 G_j gates, we would get 26 T gates, 26 H gates and 14 S gates by extrapolation, for a total of 65 gates.

Plato implementation of gate sequence approximations

We have implemented a combination of Fowler's method and the more recent single-qubit *"normal form"* representation by Matsumoto and Amano [41,42] in Haskell, to find approximating sequences. With this Haskell implementation, for example, we found an approximating sequence for $R_{\pi/256}$ with distance $\delta = 3.6 \times 10^{-4}$, and with sequence length 74:

$$\begin{aligned} R_{\pi/256} \approx\ & SHTHTHTSHTHTSHTHTSHTSHTSHT \\ & \times HTHTHTSHTSHTSHTHTSHTHTSH \\ & \times THTSHTSHTSHTHTHTHTSHTHSS. \end{aligned} \tag{27}$$

This sequence consists of 28 (37.8%) T gates, 29 (39.2%) H gates, and 17 (23%) S gates. Smaller rotations tend to need longer sequences to reach the distance threshold δ and/or improve on the identity as best approximation. Because our search algorithm used to find the approximating sequences, like Fowler's method, has exponential running time, finding a specific sequence to approximate a specific arbitrary rotation is not always feasible. Recent progress on this topic aiming at optimal-depth single-qubit rotation decompositions [47–52] highlights the importance of this problem for quantum computing.

For our QLSA LRE we have made the following simple (and rather pessimistic) assumption: namely, that any arbitrary single-qubit rotation gate (a large number of such gates, with various angles of rotation, occurs in the implementation of QLSA) can be approximated using approx. 100 fault-tolerant gates from the standard set $\{X, Y, Z, H, S, T\}$ while also achieving the desired level of algorithmic accuracy ($\varepsilon = 0.01$). This approximation turned out to be indeed fairly conservative for all rotation gates we had found specific sequences for. Following the above stable relative fractions of approximately 40% T gates, 40% H gates, and 20% S gates in the approximating sequences found, we roughly assume that, *on average*, each arbitrary rotation in fact consists of 40 T gates, 40 H gates and 20 S gates.

Taking an implementation accuracy $\varepsilon = 0.01$ for each single-qubit rotation gate is not sufficient to guarantee accuracy $\varepsilon = 0.01$ for the entire algorithm. To achieve the latter, we would typically require a much smaller target accuracy for the implementation of single-qubit rotation gates. If the entire algorithm consists of n_R single-qubit rotations, requiring a target accuracy $\varepsilon' = \varepsilon/n_R$ for each rotation would be an obvious choice. This is a fairly conservative error bound though, presuming that all rotations are performed in a sequence, with errors in different rotations adding up, never canceling each other out, and disregarding any parallelism in their implementations. However, errors may cancel each other out during the mostly sequential implementation of the gates. The LRE analysis of the bare algorithm excluding oracle resources revealed roughly $n_R \approx 10^{23}$ single-qubit rotations (with nontrivial angles of rotation), most of which have to be performed sequentially, as implied by the distinct lack of parallelism in the design of QLSA. According to Fowler's analysis, the number of standard gates needed on average to implement (decompose) a single-qubit rotation with accuracy $\varepsilon' = \varepsilon/n_R$ is approximately: $l = \frac{\log_{10}(\varepsilon/n_R)/0.292}{-0.051}$, cf. Eq. (26). Inserting the values

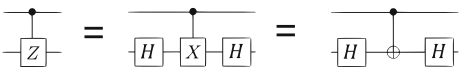

Fig. 29 Controlled-Z gate in terms of standard gates

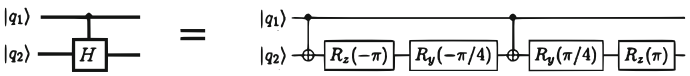

Fig. 30 Implementation of controlled-H gate in terms of CNOTs and single-qubit rotations

$n_R \approx 10^{23}$ and $\varepsilon = 0.01$ yields $l \approx 480$, which is less than by a factor 5 larger than what we assumed for our LRE analysis. Hence, while our LRE results in Table 2 provide gate counts for what is necessary (not sufficient) to achieve an accuracy $\varepsilon = 0.01$ for the entire algorithm, the more conservative error bound $\varepsilon' = \varepsilon/n_R$ for the target rotation accuracy (to guarantee the accuracy ε for the whole algorithm) would yield estimates for H, S, and T gates as well as T-depth that are only by a factor ∼5 larger. The overall gate count and overall circuit depth would also be increased by a slightly smaller factor close to 5.

Appendix 2: Circuits and resource estimates of lower-level subroutines and multiqubit gates employed by QLSA

Here we review some well-known circuit decompositions of various multiqubit gates in terms of the standard set of elementary gates $\{X, Y, Z, H, S, T, \text{CNOT}\}$ and their associated resource counts that have been used for our QLSA LRE analysis.

Controlled-Z gate

Controlled-Z gate can be decomposed into two H gates and one CNOT according to Fig. 29.

Controlled-H gate

Controlled-H gate can be implemented in terms of standard gates by using the circuit equality given in Fig. 30: The single-qubit rotations employed in this implementation can be further decomposed into sequences consisting only of T, S and H gates: $R_z(\pi) = T^4 = S^2 = Z$, $R_z(-\pi) = S^{\dagger 2} = Z$, $R_y(\pi/4) = SHTSHXZS$ and $R_y(-\pi/4) = S^\dagger ZXHS^\dagger T^\dagger HS^\dagger$.

W-gate

"W-gate" is a two-qubit gate whose action as well as its implementation in terms of standard gates is illustrated in Fig. 31. As described above for the "controlled-H" gate, the single-qubit rotations $R_z(\pi)$, $R_z(-\pi)$, $R_y(\pi/4)$ and $R_y(-\pi/4)$ can be further decomposed in terms of sequences consisting only of T, S and H gates.

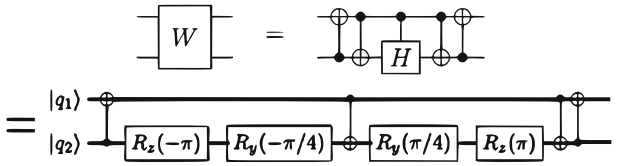

Fig. 31 Definition of the two-qubit "W-gate" and its implementation in terms of CNOTs and single-qubit rotations

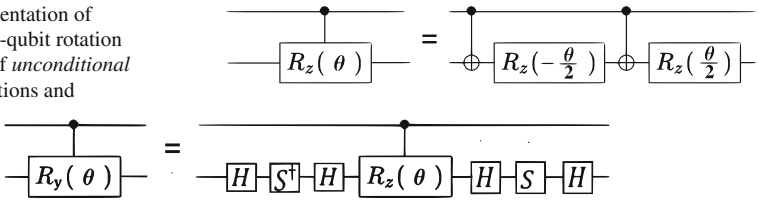

Fig. 32 Implementation of *controlled* single-qubit rotation $R_z(\theta)$ in terms of *unconditional* single-qubit rotations and CNOTs

Fig. 33 Implementation of controlled single-qubit rotation $R_y(\theta)$ in terms of controlled single-qubit rotation $R_z(\theta)$ and standard single-qubit gates

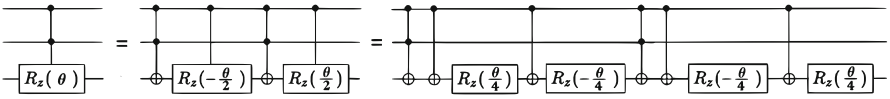

Fig. 34 Implementation of doubly controlled single-qubit rotation $R_z(\theta)$ in terms of Toffolis, CNOTs, and unconditional single-qubit rotations

Controlled rotations

Controlled single-qubit rotations $R_z(\theta)$ can be implemented in terms of CNOTs and unconditional single-qubit rotations according to circuit equality provided in Fig. 32. In the case of controlled single-qubit rotations $R_y(\theta)$ we can use the circuit identity shown in Fig. 33. A similar implementation can be derived for controlled single-qubit rotations $R_x(\theta)$. Moreover, *doubly controlled* rotations can be implemented in terms of Toffolis, CNOTs, and unconditional single-qubit rotations according to circuit equality given in Fig. 34.

Toffoli gate

Toffoli gate (essentially a CCNOT) can be implemented (cf., e.g., [1]) by a circuit using 6 CNOT gates, 1 S gate, 7 T (or $T^\dagger$) gates and 2 Hadamard gates, and having circuit depth 12, see Fig. 35.

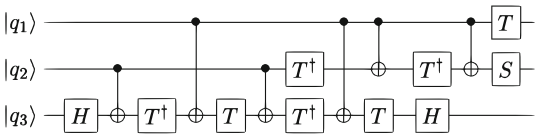

Fig. 35 Decomposition of Toffoli gate in terms of standard set of gates

Table 3 Resource requirement of multicontrolled NOT employing n control qubits and a single target qubit

Elementary resource	Resource count
Ancilla qubits	$n-2$
H gates	$2(2n-3)$
S gates	$(2n-3)$
T gates	$7(2n-3)$
CNOT gates	$6(2n-3)$
Circuit width	$n+1$
Circuit depth	$12(2n-3)$
T-depth	$6(2n-3)$
Measurements	$(n-2)$

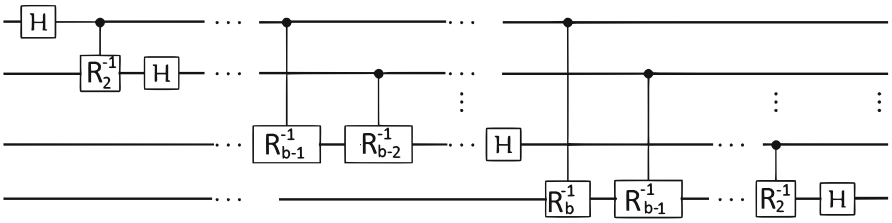

Fig. 36 Quantum circuit to implement QFT^{-1} acting on b qubits, where $R_k := \mathrm{diag}\left(1, e^{2\pi i/2^k}\right)$

Multicontrolled NOT

A multifold CNOT that is controlled by $n \geq 3$ qubits can be implemented by $2(n-2)+1$ Toffoli gates, which must be performed *sequentially*, and employing $(n-2)$ additional ancilla qubits [38]. Using the resources needed for Toffoli gates, we can infer the resource count of any multicontrolled NOT employing an arbitrary number of control qubits and a single target qubit, see Table 3.

Quantum Fourier Transform (QFT)

Both Quantum Fourier Transform (QFT) and its inverse QFT^{-1} are employed in the implementation of QLSA. QFT and its representation in terms of a quantum circuit are discussed in most introductory textbooks on quantum computation, see e.g., [1]. A circuit implementation of QFT^{-1} is shown in Fig. 36, where we use the definition $R_k := \left(\begin{smallmatrix} 1 & 0 \\ 0 & \exp(2\pi i/2^k) \end{smallmatrix}\right)$.

Here we expand on elementary resource requirements of QFT (and its inverse QFT^{-1}). Let $b \geq 2$ be the number of qubits the QFT (or its inverse) acts on, as in Fig. 36. Using the circuit decomposition rule for controlled rotations discussed in Appendix "Controlled rotations," we can derive the circuit identity shown in Fig. 37. Using this circuit identity rule, we can express the logical resource requirements in terms of standard gates and unconditional R_k gates, see Table 4. The latter can then

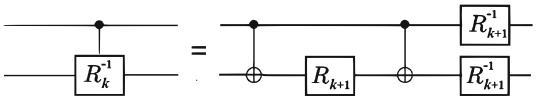

Fig. 37 Implementing controlled-R_k gates from circuit in Fig. 36 in terms of CNOT's and *unconditional* R_k gates

Table 4 Resource requirement of QFT (or its inverse transformation QFT^{-1}) in terms of standard gates and *unconditional* R_k gates

Elementary resource	Resource count
H gates	b
Unconditional R_k (or R_k^{-1}) where $k = 3, \ldots, b+1$ and $R_k := \left(\begin{smallmatrix} 1 & 0 \\ 0 & \exp(2\pi i/2^k) \end{smallmatrix}\right)$	$3(b-k+2)$ for particular k $\frac{3}{2}b(b-1)$ in total
CNOT gates	$b(b-1)$
Circuit width	b
Circuit depth	$b^2 + 2\sum_{j=3}^{b+1}\sum_{k=3}^{j}$ c-depth(R_k)
T-depth	$2\sum_{j=3}^{b+1}\sum_{k=3}^{j}$ T-depth(R_k)

The number of qubits involved in the transformation is denoted by b. The unconditional R_k (or R_k^{-1}) gates can be approximated by sequences consisting only of fault-tolerant gates T, S and H

be implemented in terms of approximating sequences consisting only of fault-tolerant gates from the set $\{X, Y, Z, H, S, T\}$, as discussed in "Appendix 1."

Controlled phase: C-Phase(c; ϕ_0, f)

The task of the controlled-phase C-Phase($\mathbf{c}$; ϕ_0, f), which is a lower-level algorithmic building block used in the implementations of the higher-level subroutines "StatePrep_**b**," "StatePrep_**R**" and "HamiltonianSimulation" (see Fig. 2), is to apply a phase shift to a signed n-qubit input register $\mathbf{c}$, whereby the applied phase is controlled by $\mathbf{c}$ itself:

$$|\mathbf{c}\rangle \to e^{-(-i)^f \theta Z/2} |\mathbf{c}\rangle\,, \text{ with } \theta = \sum_{i=0}^{n-2} 2^i \phi_0 \delta_{\mathbf{c}[i],1}. \tag{28}$$

Note, that the first $\mathbf{c}$-register qubit $\mathbf{c}[0]$ signifies the least significant bit corresponding to the minimum phase shift ϕ_0, whereas the qubit $\mathbf{c}[n-2]$ determines the most significant bit. Moreover, the last $\mathbf{c}$-register qubit $\mathbf{c}[n-1]$ controls the sign of the applied phase. To implement inverse operations, it is conditionally flipped by a classical integer flag $f \in \{0, 1\}$; for $f = 1$ the phase should be inverted. The quantum circuit is provided in Fig. 38.

When employed as part of the subroutine $M = \mathrm{Hmag}(\mathbf{x}, \mathbf{y}, \mathrm{m}, \phi_0)$, the controlled-phase C-Phase($\mathbf{c}$; ϕ_0, f) is *in addition* to be controlled by a single-qubit $\mathbf{t}[j]$ that is part of the n_1-qubit HS control register $\mathbf{t}$, see Figs. 19 and 39. For the LREs of C-Phase($\mathbf{c}$; ϕ_0, f) and C-Phase that is further controlled by a single-qubit $\mathbf{t}[j]$, we utilized the circuit decomposition rules discussed in the previous appendix sections. In particular, we used the rough (and rather conservative) assumption that, on average,

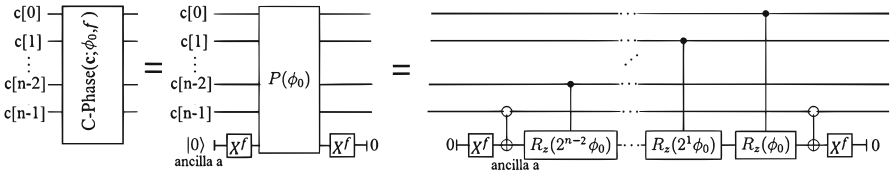

Fig. 38 Quantum circuit to implement the *controlled phase* subroutine C-Phase($\mathbf{c}$; ϕ_0, f)

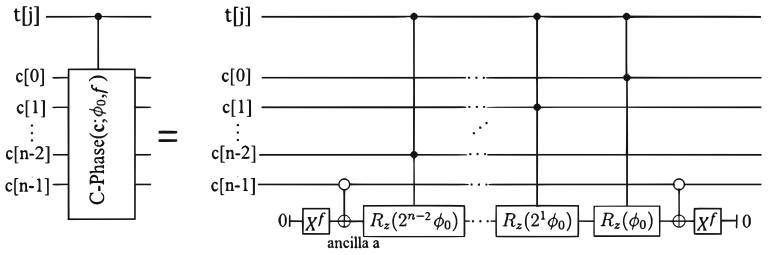

Fig. 39 Quantum circuit to implement the controlled-phase subroutine C-Phase($\mathbf{c}$; ϕ_0, f), that is *in addition* further controlled by a single-qubit control register $\mathbf{t}[j]$

Table 5 Resource estimates for the *unconditional* C-Phase($\mathbf{c}$; ϕ_0, f) subroutine implemented by circuit given in Fig. 38, where $\mathbf{c}$ is an n-qubit register with $n \in \mathbb{N}$, and $f \in \{0, 1\}$ a classical integer flag

Elementary resource	Resource count
Ancilla qubits	1
H gates	$80(n-1)$
S gates	$40(n-1)$
T gates	$80(n-1)$
X gates	$4+2f$
CNOT gates	$2n$
Circuit width	$n+1$
Circuit depth	$202(n-1)+6$
T-depth	$80(n-1)$
Measurements	1

every (unconditional) single-qubit rotation gate can be approximated by sequences of approx. 100 fault-tolerant gates with each sequence roughly consisting of 40 T gates, 40 H gates and 20 S gates, see "Appendix 1." The LREs of unconditional C-Phase and conditional C-Phase are summarized in Tables 5 and 6.

Controlled-RotY: C-RotY($\mathbf{c}$, $\mathbf{t}$; ϕ_0, f)

The task of the subroutine C-RotY($\mathbf{c}$, $\mathbf{t}$; ϕ_0, f), which is used in the implementation of higher-level subroutines "StatePrep_**b**," "StatePrep_**R**" and "Solve_x," is to apply a single-qubit rotation $R_y(\theta)$ to a *single*-qubit target register $\mathbf{t}$, where the angle of rotation θ is controlled by a signed n-qubit input register $\mathbf{c}$:

Table 6 Resource estimates for the *conditional* C-Phase($\mathbf{c}$; ϕ_0, f) subroutine implemented by circuit in Fig. 39, where $\mathbf{c}$ is an n-qubit register with $n \in \mathbb{N}$, and $f \in \{0, 1\}$ a classical integer flag

Elementary resource	Resource count
Ancilla qubits	1
H gates	$164(n-1)$
S gates	$82(n-1)$
T gates	$174(n-1)$
X gates	$4+2f$
CNOT gates	$16(n-1)+2$
Circuit width	$n+2$
Circuit depth	$436(n-1)+6$
T-depth	$174(n-1)$
Measurements	1

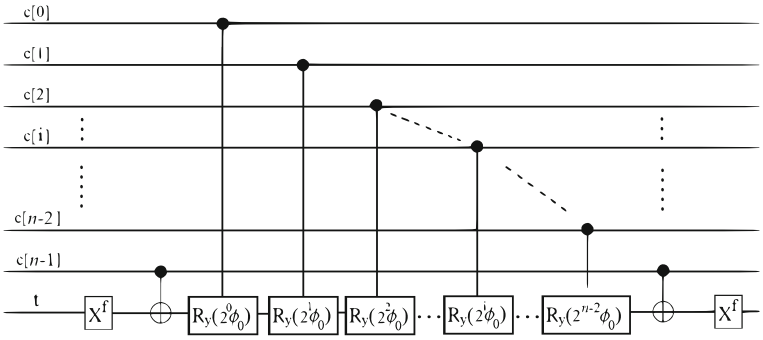

Fig. 40 Quantum circuit to implement the subroutine C-RotY($\mathbf{c}, \mathbf{t}$; ϕ_0, f), employing an n-qubit control register $\mathbf{c}$ and a single-qubit target register $\mathbf{t}$. The classical integer flag $f \in \{0, 1\}$ facilitates inverse transformations

$$|\mathbf{t}\rangle \to e^{-(-i)^f \theta Y/2} |\mathbf{t}\rangle \quad \text{with} \quad \theta = \sum_{i=0}^{n-2} 2^i \phi_0 \delta_{\mathbf{c}[i],1}. \tag{29}$$

The first $\mathbf{c}$-register qubit $\mathbf{c}[0]$ signifies the least significant bit corresponding to the minimum angle of rotation ϕ_0, whereas the qubit $\mathbf{c}[n-2]$ determines the most significant bit. The sign of the applied rotation is controlled by the last $\mathbf{c}$-register qubit $\mathbf{c}[n-1]$. In addition, it is conditionally flipped by a classical integer flag $f \in \{0, 1\}$ to enable straightforward inverse operations. The quantum circuit is provided in Fig. 40. For the LRE of subroutine C-RotY($\mathbf{c}, \mathbf{t}$; ϕ_0, f), we utilized the circuit decomposition rules discussed in the previous appendix sections; our estimates are summarized in Table 7.

Appendix 3: Resource estimates for the oracles

Below we report our LRE results for some representative oracle queries; all other oracle queries have similar resource counts. These results depend on several choices: the

Table 7 Resource estimates for C-RotY($\mathbf{c}, \mathbf{t}; \phi_0, f$) subroutine, whose quantum circuit is shown in Fig. 40, where $\mathbf{c}$ is an n-qubit input register with $n \in \mathbb{N}$, and $f \in \{0, 1\}$

Elementary resource	Resource count
Ancilla qubits	0
H gates	$84(n-1)$
S gates	$42(n-1)$
T gates	$80(n-1)$
X gates	$2f$
CNOT gates	$2n$
Circuit width	$n+1$
Circuit depth	$202(n-1)+2f$
T-depth	$80(n-1)$
Measurements	1

internal representation for real and integer numbers, the details of the linear-system problem definition, and the method for generating oracles. As for the internal representation of numbers, since every single operation had to be built from scratch, we used *fixed-point representation*. Compared to a floating-point representation, it is simpler and therefore generates smaller circuits. Regarding the details of the linear-system problem definition, they constitute the core data of this particular implementation of QLSA; provided in the GFI, we made no effort to modify them. Finally, the oracles were generated with an automated tool, turning a classical description of an algorithm into a reversible quantum circuit. We made this choice because we felt that it was the most natural (and practical) solution for the particular kind of oracles we were dealing with: general functions over real and complex numbers.

Quipper automatically generates recursive decompositions of oracles down to the level of gates such as initialization, termination, etc. and controlled-NOTs (by at most one or two wires, each on either true or false). The rules for decomposing these gates into the standard-basis gates H, S, T, and X, and calculating circuit depths and T-depths are included manually. Our rules for the depths are very conservative: we assume sequential executions unless we know better strategies. Indeed, optimal-depth decompositions are known only for fairly small gates, such as e.g., the Toffoli gate. Hence we expect over-estimates both for circuit- and T-depths.[20] These recursive gate-decomposition rules are coded in the symbolic programming software Mathematica for computing the final estimates.

Oracle A returns either the magnitude (`argflag=False`) or the phase (`argflag=True`) of the coupling weight and the connected node index at the chosen matrix-decomposition-band index (from 1 to $N_b = 9$). As there are many combinations, we will show a representative sample and will draw conclusions from them. As is evident from Table 8 that the estimates for different bands in the

[20] As discussed previously, our circuit implementations of oracles are essentially the *trace of execution* of a classical program of an algorithm. Because the algorithms we used are purely sequential, the corresponding quantum circuits are not easily parallelizable on a global scale. The only possible optimizations are purely local. We therefore conclude that our computed circuit- and T-depth values are over-estimates by some unknown small factor wrt. optimal-depth values.

Table 8 Resource estimation for Oracle A with `argflag = False`, for various bands = 1, 3, and 5

Elementary resource	Resource count for band = 1	Resource count for band = 3	Resource count for band = 5
Ancilla qubits	$4{,}779{,}020 \simeq 4.78 \times 10^6$	$4{,}780{,}967 \simeq 4.78 \times 10^6$	$4{,}775{,}909 \simeq 4.78 \times 10^6$
H gates	$36{,}206{,}376 \simeq 36.2 \times 10^6$	$36{,}205{,}072 \simeq 36.2 \times 10^6$	$36{,}183{,}272 \simeq 36.2 \times 10^6$
S gates	$18{,}103{,}188 \simeq 18.1 \times 10^6$	$18{,}103{,}036 \simeq 18.1 \times 10^6$	$18{,}091{,}636 \simeq 18.1 \times 10^6$
T gates	$126{,}722{,}316 \simeq 126.7 \times 10^6$	$126{,}721{,}252 \simeq 126.7 \times 10^6$	$126{,}641{,}452 \simeq 126.6 \times 10^6$
X gates	$24{,}314{,}146 \simeq 24.3 \times 10^6$	$24{,}313{,}698 \simeq 24.3 \times 10^6$	$24{,}302{,}823 \simeq 24.3 \times 10^6$
CNOT gates	$116{,}377{,}976 \simeq 116.38 \times 10^6$	$116{,}382{,}690 \simeq 116.38 \times 10^6$	$116{,}305{,}510 \simeq 116.31 \times 10^6$
Circuit depth	$248{,}096{,}178 \simeq 248.1 \times 10^6$	$248{,}099{,}532 \simeq 248.1 \times 10^6$	$247{,}943{,}077 \simeq 247.9 \times 10^6$
T-depth	$108{,}619{,}128 \simeq 108.6 \times 10^6$	$108{,}618{,}216 \simeq 108.6 \times 10^6$	$108{,}549{,}816 \simeq 108.6 \times 10^6$
Measurements	$4{,}779{,}020 \simeq 4.78 \times 10^6$	$4{,}780{,}967 \simeq 4.78 \times 10^6$	$4{,}775{,}909 \simeq 4.78 \times 10^6$

Table 9 Representative resource estimation for Oracle A with `argflag = False` and `argflag = True`

Elementary resource	`argFlag = False`	`argFlag = True`
Ancilla qubits	4.78×10^6	1.29×10^7
H gates	3.62×10^7	1.33×10^8
S gates	1.81×10^7	6.66×10^7
T gates	1.27×10^8	4.66×10^8
X gates	2.43×10^7	7.64×10^7
CNOT gates	1.16×10^8	4.13×10^8
Circuit depth	2.48×10^8	8.87×10^8
T-depth	1.09×10^8	4.0×10^8
Measurements	4.78×10^6	12.9×10^6

Table 10 Resource estimation for Oracle b and Oracle R

Elementary resource	Resource count, Oracle b	Resource count, Oracle R
Ancilla qubits	$204{,}765{,}119 \simeq 2.1 \times 10^8$	$110{,}576{,}558 \simeq 1.1 \times 10^8$
H gates	$1{,}641{,}762{,}800 \simeq 1.6 \times 10^9$	$888{,}704{,}520 \simeq 8.9 \times 10^8$
S gates	$820{,}881{,}400 \simeq 8.2 \times 10^8$	$444{,}352{,}260 \simeq 4.4 \times 10^8$
T gates	$5{,}746{,}169{,}800 \simeq 5.7 \times 10^9$	$3{,}110{,}465{,}820 \simeq 3.1 \times 10^9$
X gates	$1{,}075{,}933{,}016 \simeq 1.1 \times 10^9$	$582{,}282{,}144 \simeq 5.8 \times 10^8$
CNOT gates	$5{,}241{,}180{,}190 \simeq 5.2 \times 10^9$	$2{,}836{,}515{,}650 \simeq 2.8 \times 10^9$
Circuit depth	$11{,}192{,}585{,}310 \simeq 11 \times 10^9$	$6{,}057{,}980{,}506 \simeq 6.06 \times 10^9$
T-depth	$4{,}925{,}288{,}400 \simeq 4.9 \times 10^9$	$2{,}666{,}113{,}560 \simeq 2.67 \times 10^9$
Measurements	$204{,}765{,}119 \simeq 2.0 \times 10^8$	$110{,}576{,}558 \simeq 1.1 \times 10^8$

`argflag = False` cases all agree to the subone-percent level, or to three significant figures, with the exception of the number of qubits which only agree to within about three percents of each other, or to two significant figures. Therefore anyone of them can be taken as a representative for all `argflag = False` Oracle A resource estimates and a representative table is also presented. Similar phenomenon is true for all the `argflag = True` cases and only a representative table is presented for them. As gate decompositions used are to the basis-gate level, the number of ancillas and measurements should agree in every case, each with individual band index and argflag. This is indeed true in all cases for which we have performed resource counting. The two representative tables for `argflag = False` and `argflag = True` are presented in Table 9. Finally, the resource counts for Oracle r and for Oracle b are done similarly: Quipper gives logical resource estimates, then recursive gate-decomposition rules are coded in the symbolic computing software Mathematica for computing the final estimates presented in Table 10.

One may wonder why our oracle implementations require such a huge number of auxiliary qubits and measurements—namely, up to $\sim 10^8$ ancilla qubits and measurements for a problem size $N \approx 3 \times 10^8$. This indeed is a feature of our low-level implementation of the *irreversible-to-reversible* transformations that is similar to the way "logical reversibility of computation" was proposed by Bennett in [53]. In essence, to ensure that the run of the entire computation can be unwound, the result of each of its elementary subcomputations is stored in an auxiliary qubit. When the final result has been computed, it is copied into a fresh quantum register, and the entire computation is reversed, with every subcomputation undone along the way, and the initial values "0" of the intermediate auxiliary qubits restored and verified by a measurement. The number of auxiliary qubits required is therefore directly proportional to the number of elementary computational steps, and thus to the number of gates in the oracle. And the number of measurements needed to ensure reversibility of computation equals the number of ancilla qubits. One might argue that such an implementation is unnecessarily verbose. While we agree that there may be more efficient implementations (e.g., by using some known efficient adders when performing addition), our proposed implementation is arguably not so inefficient, in the sense that the size of the circuit (and therefore also the number of auxiliary qubits) is directly proportional (and not, say, exponential) to the length of the classical computation that would compute the data. In particular, the size of the circuit for the oracle computing an element of the matrix A is linear in the number of bits required to store the size of the matrix. Hence, the Bennett construction we follow for our oracle implementations has good "theoretic properties" in the worst case. However, the overhead for the implementation of an arbitrary oracle is still considerable. Yet it is very useful as a first baseline resource estimate. There is scope for improvement, both from software tools and from better algorithm design using reversible gates.

References

1. Nielsen, M.A., Chuang, I.L.: Quantum Computing and Quantum Information. Cambridge University Press, Cambridge (2000)
2. Jordan, S.: Quantum Algorithm Zoo (2013). URL: http://math.nist.gov/quantum/zoo/
3. Harrow, A.W., Hassidim, A., Lloyd, S.: Quantum algorithm for linear systems of equations. Phys. Rev. Lett. **103**, 150502 (2009)
4. Ambainis, A.: Variable time amplitude amplification and a faster quantum algorithm for solving systems of linear equations. arXiv:1010.4458 (2010)
5. Clader, B.D., Jacobs, B.C., Sprouse, C.R.: Preconditioned quantum linear system algorithm. Phys. Rev. Lett. **110**, 250504 (2013)
6. Luis, A., Peřina, J.: Optimum phase-shift estimation and the quantum description of the phase difference. Phys. Rev. A **54**, 4564 (1996)
7. Cleve, R., Ekert, A., Macchiavello, C., Mosca, M.: Quantum Algorithms Revisited. arXiv:quant-ph/9708016 (1997)
8. Berry, D.W., Ahokas, G., Cleve, R., Sanders, B.C.: Efficient quantum algorithms for simulating sparse Hamiltonians. Commun. Math. Phys. **270**(2), 359 (2007)
9. Wiebe, N., Braun, D., Lloyd, S.: Quantum Data Fitting. Phys. Rev. Lett. **109**, 050505 (2012)
10. Berry, D.W.: High-order quantum algorithm for solving linear differential equations. J. Phys. A Math. Theor. **47**, 105301 (2014)
11. Lloyd, S., Mohseni, M., Rebentrost, P.: Quantum algorithms for supervised and unsupervised machine learning. arXiv:1307.0411 (2013)

12. Barz, S., Kassal, I., Ringbauer, M., Lipp, Y., Dakic, B., Aspuru-Guzik, A., Walther, P.: Solving systems of linear equations on a quantum computer. Sci. Rep. **4**, 6115 (2014). doi:10.1038/srep06115
13. Cai, X.D., Weedbrook, C., Su, Z.E., Chen, M.C., Gu, M.J.Z.M., Li, L., Liu, N.L., Lu, C.Y., Pan, J.W.: Experimental quantum computing to solve systems of linear equations. Phys. Rev. Lett. **110**, 230501 (2013)
14. Green, A., Lumsdaine, P.L., Ross, N.J., Selinger, P., Valiron, B.: Quipper: a scalable quantum programming language. In: Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI'13, pp. 333–342 (2013)
15. Green, A., Lumsdaine, P.L., Ross, N.J., Selinger, P., Valiron, B.: An introduction to quantum programming in Quipper. In: Proceedings of the 5th International Conference on Reversible Computation, Lecture Notes in Computer Science, vol. 7948 , Lecture Notes in Computer Science, vol. 7948, pp. 110–124 (2013)
16. Intelligence Advanced Research Projects Activity (IARPA). Quantum Computer Science (QCS) Program (2010). URL http://www.iarpa.gov/index.php/research-programs/qcs
17. Intelligence Advanced Research Projects Activity (IARPA). Quantum Computer Science (QCS) Program Broad Agency Announcement (BAA) (April 2010). URL http://www.iarpa.gov/index.php/research-programs/qcs/baa
18. The Quipper Language (2013). URL http://www.mathstat.dal.ca/~selinger/quipper/
19. The Quipper System (2013). URL http://www.mathstat.dal.ca/~selinger/quipper/doc/
20. Shewchuk, J.R.: An Introduction to the Conjugate Gradient Method Without the Agonizing Pain. (Technical Report CMU-CS-94-125 School of Computer Science, Carnegie Mellon University, Pittsburgh, Pennsylvania (1994))
21. Saad, Y.: Iterative Methods for Sparse Linear Systems. Society for Industrial and Applied Mathematics, Philadelphia (2003)
22. Brassard, G., Hoyer, P., Mosca, M., Tapp, A.: Quantum amplitude amplification and estimation. In: Quantum Computation and Quantum Information, vol. 305 (AMS Contemporary Mathematics, 2002), pp. 53–74 (2002)
23. Jin, J.M.: The Finite Element Method in Electromagnetics. Wiley, New York (2002)
24. Chatterjee, A., Jin, J.M., Volakis, J.L.: Edge-based finite elements and vector ABCs applied to 3D scattering. IEEE Trans. Antennas Propagat. **41**, 221 (1993)
25. Trotter, H.: On the product of semi-groups of operators. In: Proceedings of the American Mathematical Society, vol. 10, pp. 545–551 (1959)
26. Suzuki, M.: Fractal decomposition of exponential operators with applications to many-body theories and Monte-Carlo simulations. Phys. Lett. A **146**, 319 (1990)
27. Brenner, S.C., Scott, L.R.: The Mathematical Theory of Finite Element Methods. Springer, New York (2008)
28. Bank, R.E., Scott, L.R.: On the conditioning of finite element equations with highly refined meshes. SIAM J. Numer. Anal. **26**(6), 1383 (1989)
29. Layton, W.: High-accuracy finite-element methods for positive symmetric systems. Comput. Math. Appl. **12A**(4/5), 565 (1986)
30. Childs, A.M., Cleve, R., Deotto, E., Farhi, E., Gutmann, S., Spielman, D.A.: Exponential algorithmic speedup by a quantum walk. In: Proceedings of the Thirty-Fifth Annual ACM Symposium on Theory of Computing, STOC'03 (New York, NY, USA, 2003), pp. 59–68 (2003)
31. Ömer, B.: Quantum Programming in QCL. Master's thesis, Institute of Information Systems, Technical University of Vienna (2000)
32. Claessen, K.: Embedded Languages for Describing and Verifying Hardware. Ph.D. thesis, Chalmers University of Technology and Göteborg University (2001)
33. Altenkirch, T., Green, A.S.: The quantum IO monad. In: Gay, S., Mackie, I. (eds.) Semantic Techniques in Quantum Computation, pp. 173–205. Cambridge University Press, Cambridge (2009)
34. Selinger, P., Valiron, B.: A lambda calculus for quantum computation with classical control. Math. Struct. Comput. Sci. **16**(3), 527 (2006)
35. Selinger, P., Valiron, B.: Quantum lambda calculus. In: Gay, S., Mackie, I. (eds.) Semantic Techniques in Quantum Computation, pp. 135–172. Cambridge University Press, Cambridge (2009)
36. Laundauer, R.: Irreversibility and heat generation in the computing process. IBM J. Res. Dev. **5**, 261 (1961)
37. Draper, T.G., Kutin, S.A., Rains, E.M., Svore, K.M.: A logarithmic-depth quantum carry-lookahead adder. Quantum Inf. Comput. **6**, 351 (2006)

38. Mermin, N.D.: Quantum Computer Science: An Introduction. Cambridge University Press, Cambridge (2007)
39. Fowler, A.G.: Towards Large-Scale Quantum Computation. Ph.D. thesis, arXiv:quant-ph/0506126 (2005)
40. Fowler, A.G.: Constructing arbitrary Steane code single logical qubit fault-tolerant gates. Quantum Inf. Comput. **11**, 867 (2011)
41. Matsumoto, K., Amano, K.: Representation of Quantum Circuits with Clifford and $\pi/8$ Gates. arXiv:0806.3834 (2008)
42. Giles, B., Selinger, P.: Remarks on Matsumoto and Amano's normal form for single-qubit Clifford+T operators. arXiv:1312.6584 (2013)
43. Berry, D.W., Cleve, R., Somma, R.D.: Exponential improvement in precision for Hamiltonian-evolution simulation. arXiv:1308.5424v3 (2013)
44. Berry, D.W., Childs, A.M., Cleve, R., Kothari, R., Somma, R.D.: Exponential improvement in precision for simulating sparse Hamiltonians. In: Proceedings of the 46th ACM Symposium on Theory of Computing (STOC 2014), pp. 283–292 (2014)
45. Berry, D.W., Childs, A.M., Kothari, R.: Hamiltonian simulation with nearly optimal dependence on all parameters. In: Proceedings of the 56th IEEE Symposium on Foundations of Computer Science (FOCS 2015), pp. 792-809 (2015)
46. Pham, T.T., Meter, R.V., Horsman, C.: Optimization of the Solovay–Kitaev algorithm. Phys. Rev. A **87**, 052332 (2013)
47. Giles, B., Selinger, P.: Exact synthesis of multiqubit Clifford+T circuits. Phys. Rev. A **87**, 032332 (2013)
48. Bocharov, A., Gurevich, Y., Svore, K.M.: Efficient decomposition of single-qubit gates into V basis circuits. Phys. Rev. A **88**(012313), 13 (2013)
49. Selinger, P.: Optimal ancilla-free Clifford+T approximation of Z-rotations. arXiv:1403.2975 (2014)
50. Kliuchnikov, V., Maslov, D., Mosca, M.: Asymptotically optimal approximation of single qubit unitaries by Clifford and T circuits using a constant number of ancillary qubits. Phys. Rev. Lett. **110**(190502), 5 (2013)
51. Kliuchnikov, V., Maslov, D., Mosca, M.: Fast and efficient exact synthesis of single qubit unitaries generated by Clifford and T gates. Quantum Inf. Comput. **13**(7–8), 607 (2013)
52. Selinger, P.: Efficient Clifford+T approximation of single-qubit operators. arXiv:1212.6253 (2012)
53. Bennett, C.H.: Logical reversibility of computation. IBM J. Res. Dev. **17**(6), 525 (1973)