Practical Quantum Computing: solving the wave equation using a quantum approach

Adrien Suau,[1, 2, *] Gabriel Staffelbach,[1] and Henri Calandra[3]

[1]*CERFACS, 42 Avenue Gaspard Coriolis, 31057 Toulouse, France*
[2]*LIRMM, University of Montpellier, 161 rue Ada, 34095 Montpellier, France*
[3]*TOTAL SA, 2 Avenue de Vignancour, 64000 Pau, France*

(Dated: November 4, 2021)

In the last years, several quantum algorithms that try to address the problem of partial differential equation solving have been devised. On one side, "direct" quantum algorithms that aim at encoding the solution of the PDE by executing one large quantum circuit. On the other side, variational algorithms that approximate the solution of the PDE by executing several small quantum circuits and making profit of classical optimisers. In this work we propose an experimental study of the costs (in terms of gate number and execution time on a idealised hardware created from realistic gate data) associated with one of the "direct" quantum algorithm: the wave equation solver devised in [PCS. Costa, S. Jordan, A. Ostrander, *Phys. Rev. A* 99, 012323, 2019]. We show that our implementation of the quantum wave equation solver agrees with the theoretical big-O complexity of the algorithm. We also explain in great details the implementation steps and discuss some possibilities of improvements. Finally, our implementation proves experimentally that some PDE can be solved on a quantum computer, even if the direct quantum algorithm chosen will require error-corrected quantum chips, which are not believed to be available in the short-term.

I. INTRODUCTION

Quantum computing has drawn a lot of attention in the last few years, following the successive announcements from several world-wide companies about the implementation of quantum hardware with an increasing number of qubits or reduced error rates [4, 8, 9, 12, 52].

Along with the hardware improvement, new quantum algorithms were discovered, yielding potential quantum speed-up and applications in various fields such as quantum chemistry [23], linear algebra [22, 38, 40, 41, 55, 66] or optimisation [35, 42, 43]. Recent works even show that differential equations may be solved by using a quantum computer [11, 16, 21, 25, 26, 37, 46, 51, 58, 62, 65]. But despite the large number of algorithms available, it is hard to find an actual implementation of a quantum differential equation solver, Hamiltonian simulation being the unique exception by solving the time-dependant Schrödinger equation.

In this work, we present and analyse a quantum wave equation solver we implemented from scratch according to the algorithm depicted in [32]. During the solver implementation, we had to look for a Hamiltonian Simulation procedure. The implementations we found being too restricted, we decided to implement our own Hamiltonian Simulation procedure, which will also be analysed.

To the best of our knowledge, this work is the first to analyse experimentally the characteristics of a quantum PDE solver. Such a study has already been performed on the HHL algorithm, in [54]. We checked that the practical implementation agrees with the theoretical asymptotic complexities on several quantities of interest such as the total gate count with respect to the number of discretisation points used or the precision, the number of qubits required versus the number of discretisation points used to approximate the solution or precision of the solution when compared to a classical finite-difference solver. Finally, we verified that the execution time of the generated quantum circuit on today's accessible quantum hardware was still following the theoretical asymptotic complexities devised for the total gate count. Quantum hardware data were extracted from IBM Q chips.

We show experimentally that it is possible to solve the 1-dimensional wave equation on a quantum computer with a time-complexity that grows as $\mathcal{O}\left(N_d^{3/2} \log{(N_d)}^2\right)$ where N_d is the number of discretisation points used to approximate the solution. But even if the asymptotic scaling is better than classical algorithms, we found out that the constants hidden in the big-O notation were huge enough to make the solver less efficient than classical solvers for reasonable discretisation sizes.

* adrien.suau@cerfacs.fr

II. PROBLEM CONSIDERED

We consider a simplified version of the wave equation on the 1-dimensional line $[0, 1]$ where the propagation speed c is constant and equal to 1. This equation can be written as

$$\frac{\partial^2}{\partial t^2}\phi(x, t) = \frac{\partial^2}{\partial x^2}\phi(x, t). \tag{1}$$

Moreover, we only consider solving eq. (1) with the Dirichlet boundary conditions

$$\frac{\partial}{\partial x}\phi(0, t) = \frac{\partial}{\partial x}\phi(1, t) = 0. \tag{2}$$

No assumption is made on initial speed $\phi(x, 0)$ and initial velocity $\frac{\partial \phi}{\partial t}(x, 0)$.

The resolution of this simplified wave equation on a quantum computer is an appealing problem for the first implementation of a PDE solver for several reasons. First, the wave equation is a well-known and intensively studied problem for which a lot of theoretical results have been verified. Secondly, even-though it is a relatively simple PDE, the wave equation can be used to solve some interesting problems such as seismic imaging [13, 14]. Finally, the theoretical implementation of a quantum wave equation solver has already been studied in [32].

In this paper, we present the complete implementation of a 1-dimensional wave equation solver using quantum technologies based on `qat` library. To the best of our knowledge, this work is the first to consider the implementation of an entire PDE solver that can run on a quantum computer. Specifically, we explain all the implementation details of the solver from the mathematical theory to the actual quantum circuit used. The characteristics of the solver are then discussed and analysed, such as the estimated gate count and estimated execution time on real quantum hardware. We show that the implementation follows the theoretical asymptotic behaviours devised in [32]. Moreover, the wave equation solver algorithm relies critically on an efficient implementation of a Hamiltonian simulation algorithm, which we have also implemented and analysed thoroughly.

III. IMPLEMENTATION

The algorithm used to solve the wave equation is explained in [32] and uses a Hamiltonian simulation procedure. Costa et al. chose the Hamiltonian simulation algorithm described in [20] for its nearly optimal theoretical asymptotic behaviour. We privileged instead the Hamiltonian simulation procedure explained in [10, 17] for its good experimental results based on [27] and its simpler implementation (detailed in appendix A).

The code has been written using `qat`, a Python library shipped with the Quantum Learning Machine (QLM), a package developed and maintained by Atos. It has not been extensively optimized yet, which means that there is still a large room for possible improvements.

All the circuits used in this paper have been generated with a subset of `qat`'s gate set:

$$\{H, X, R_y(\theta), P_h(\theta), CP_h(\theta), CNOT, CCNOT\} \tag{3}$$

and have then been translated to the gate set

$$\{U_1(\lambda), U_2(\lambda, \phi), U_3(\lambda, \phi, \theta), CNOT\} \tag{4}$$

for U_1, U_2 and U_3 defined in Equation (7) of [31] as follow:

$$U(\lambda, \phi, \theta) = \begin{pmatrix} \cos\left(\frac{\theta}{2}\right) & -e^{i\lambda}\sin\left(\frac{\theta}{2}\right) \\ e^{i\phi}\sin\left(\frac{\theta}{2}\right) & e^{i(\lambda+\phi)}\cos\left(\frac{\theta}{2}\right) \end{pmatrix} \tag{5}$$

$$U_3(\lambda, \phi, \theta) = U(\lambda, \phi, \theta) \tag{6}$$

$$U_2(\lambda, \phi) = U\left(\frac{\pi}{2}, \lambda, \phi\right) \tag{7}$$

$$U_1(\lambda) = U(0, 0, \lambda) \tag{8}$$

Note 1. The target gate set presented in eq. (4) does not correspond to the physical gate set implemented by IBM hardware (see Equation (8) of [31]). This choice is justified by the fact that IBM only provides hardware characteristics such as gate times for the gate set of eq. (4) and not for the real hardware gate set.

This implementation aims at validating in practice the theoretical asymptotic complexities of Hamiltonian simulation algorithms and providing a proof-of-concept showing that it is possible to solve a partial differential equation on a quantum computer.

A.   Sparse Hamiltonian simulation algorithm

Definition 1. *s-sparse matrix*: A s-sparse matrix with $s \in \mathbb{N}^*$ is a matrix that has at most s non-zero entries per row and per column

Definition 2. *sparse matrix*: A sparse matrix is a s-sparse matrix with $s \in \mathcal{O}\left(\log(N)\right)$, N being the size of the matrix.

In the past years, a lot of algorithms have been devised to simulate the effect of a Hamiltonian on a quantum state [17–20, 24, 29, 44, 47–50, 57]. Among all these algorithms, only few have already been implemented for specific cases [3, 7] but to the best of our knowledge no implementation is currently capable of simulating a generic sparse Hamiltonian.

The domain of application of the already existing methods being too narrow, we decided to implement our own generic sparse Hamiltonian simulation procedure. We based our work on the product-formula approach described in [10, 17]. One advantage of this approach is that product-formula based algorithms have already been thoroughly analysed both theoretically [10, 17] and practically [27, 54], and several implementations are publicly available, though restricted to Hamiltonians that can be decomposed as a sum of tensor products of Pauli matrices. Moreover, [10] provides a lot of implementation details that allowed us to go straight to the development step.

Our implementation is capable of simulating an arbitrary sparse Hamiltonian provided that it has already been decomposed into a sum of 1-sparse Hermitian matrices with either only real or only complex entries, each described by an oracle. The implementation has been validated with several automated tests and a more complex case involving the simulation of a 2-sparse Hamiltonian and described in section III B. Furthermore, it agrees perfectly with the theoretical complexities devised in [10, 17] as studied and verified in section IV.

B.   Quantum wave equation solver

Using the Hamiltonian simulation algorithm implementation, we successfully implemented a 1-dimensional wave equation solver using the algorithm described in [32] and explained in appendix B and appendix C.

For the specific case considered (eq. (1) and eq. (2)), solving the wave equation for a time T on a quantum computer boils down to simulating a 2-sparse Hamiltonian for a time $f(T)$, the function f being thoroughly described in [32] and eq. (18). The constructed quantum circuit can then be applied to a quantum state representing the initial position $\psi(x,0)$ and velocity $\frac{\partial\phi}{\partial t}(x,0)$, and will evolve this state towards a quantum state representing the final position $\phi(x,T)$ and velocity $\frac{\partial\phi}{\partial t}(x,T)$.

As for the Hamiltonian simulation procedure, the practical results we obtain from the implementation of the quantum wave equation solver seems to match the theoretical asymptotic complexities. See section IV for an analysis of the theoretical asymptotic complexities.

IV.   RESULTS

Using a simulator instead of a real quantum computer has several advantages. In terms of development process, a simulator allows the developer to perform several actions that are not possible as-is on a quantum processor such as describing a quantum gate with a unitary matrix instead of a sequence of hardware operations. Another useful operation that is possible on a quantum simulator and not currently achievable on a quantum processor is efficient generic state preparation.

Our implementation uses only standard quantum gates and does not leverage any of the simulator-only features such as quantum gates implemented from a unitary matrix. In other words, both the Hamiltonian simulation procedure and the quantum wave equation solver are "fully quantum" and are readily executable on a quantum processor, provided that it has enough qubits. As a proof, and in order to benchmark our implementation, we translated the

generated quantum circuits to IBM Q Melbourne gate-set (see eq. (4)). IBM Q Melbourne [2] is a quantum chip with 14 usable qubits made available by IBM the 23th of September, 2018.

Note 2. We chose IBM Q Melbourne mainly because, at the time of writing, it was the publicly accessible quantum chip with the larger number of qubits and so was deemed to be the closest to future quantum hardware. It is important to note that even if IBM Q Melbourne has 14 qubits, the quantum circuits constructed in this paper are not runnable because they require more qubits. Consequently, because of this hardware limitation, hardware topology has also been left apart of the study.

This allowed us to have an estimation of the number of hardware gates needed to either solve the wave equation or simulate a specific Hamiltonian on this specific hardware. Combining these numbers and the hardware gate execution time published in [6], we were able to compute a rough approximation of the time needed to solve the considered problem presented in eq. (1) and eq. (2) on this specific hardware.

A. Hamiltonian simulation

As explained in section III A, the Hamiltonian simulation algorithm implemented has been first devised in [10, 17]. A quick review of the algorithm along with implementation details can be found in appendix A. This Hamiltonian simulation procedure requires that the Hamiltonian matrix H to simulate can be decomposed as

$$H = \sum_{j=1}^{m} H_j \tag{9}$$

where each H_j is an efficiently simulable Hermitian matrix.

In our benchmark, we simulated the Hamiltonian described in eq. (B11). According to [10], real 1-sparse Hermitian matrices with only 1 or 0 entries can be simulated with $\mathcal{O}(n)$ gates and 2 calls to the oracle, n being the number of qubits the Hamiltonian H acts on. The exact gate count can be found in table I in the row `1-sparse HS`.

Let O_i be the gate complexity of the oracle implementing the ith Hermitian matrix H_i of the decomposition in eq. (9), we end up with an asymptotic complexity of $\mathcal{O}(n + O_i)$ to simulate H_i. Once again, the exact gate count is decomposed in table I.

Applying the Trotter-Suzuki product-formula of order k (see Definition definition 4 in appendix A 5 for the definition of the Trotter-Suzuki product-formula) on the quantum circuit simulating the Hermitian matrices produces a circuit of size

$$\mathcal{O}\left(5^k \sum_{i=1}^{m}(n + O_i)\right). \tag{10}$$

This circuit should finally be repeated r times in order to achieve an error of at most ϵ, with

$$r \in \mathcal{O}\left(5^k m\tau \left(\frac{m\tau}{\epsilon}\right)^{\frac{1}{2k}}\right), \tag{11}$$

and $\tau = t \max_i \|H_i\|$, t being the time for which we want to simulate the given Hamiltonian and $\|\cdot\|$ being the spectral norm [17].

Merging eq. (10) and eq. (11) gives us the complexity

$$\mathcal{O}\left(5^{2k} m\tau \left(\frac{m\tau}{\epsilon}\right)^{\frac{1}{2k}} \sum_{i=1}^{m}(n + O_i)\right). \tag{12}$$

This generic expression of the asymptotic complexity can be specialized to our benchmark case. The number of gates needed to implement the oracles is $\mathcal{O}(n^2)$ and the chosen decomposition contains $m = 2$ Hermitian matrices, each with a spectral norm of 1. Replacing the symbols in eq. (10) and eq. (11) results in the asymptotic gate complexity of

$$\mathcal{O}\left(5^k n^2\right) \tag{13}$$

for the circuit simulating $e^{-iHt/r}$ and a number

$$r \in \mathcal{O}\left(5^k t \left(\frac{t}{\epsilon}\right)^{\frac{1}{2k}}\right) \tag{14}$$

of repetitions, which lead to a total gate complexity of

$$\mathcal{O}\left(5^{2k} n^2 t \left(\frac{t}{\epsilon}\right)^{\frac{1}{2k}}\right). \tag{15}$$

In order to check that our implementation follows this theoretical asymptotic behaviour, we chose to let $k = 1$ and plotted the number of gates generated versus the three parameters that have an impact on the number of gates: the number of discretisation points N_d (fig. 1(a)), the time of simulation t (fig. 1(b)) and the precision ϵ (fig. 1(c)). The corresponding asymptotic complexity should be

$$\mathcal{O}\left(n^2 \frac{t^{3/2}}{\sqrt{\epsilon}}\right) = \mathcal{O}\left(\log_2\left(N_d\right)^2 \frac{t^{3/2}}{\sqrt{\epsilon}}\right). \tag{16}$$

A small discrepency can be observed in fig. 1(a): the theoretical asymptotic number of gates is $\mathcal{O}\left(\log_2\left(N\right)^2\right)$ but the experimental values seem better fitted with an asymptotic behaviour of $\mathcal{O}\left(\log_2\left(N\right)^{7/4}\right)$. This may be caused by the asymptotic regime not being reached yet.

B.   Wave equation solver

The first characteristic of the wave equation solver that needs to be checked is its validity: is the quantum wave equation solver capable of solving accurately the wave equation as described in eq. (1) and eq. (2)?

To check the validity of the solver, we used `qat` simulators and Atos QLM to simulate the quantum program generated to solve the wave equation with different values for the number of discretisation points N_d, for the physical time t and for the precision ϵ. fig. 3 shows the classical solution versus the quantum solution and the absolute error between the two solutions for $N_d = 32$, $t = 0.4$ and $\epsilon = 10^{-3}$. The solution obtained by the quantum solver is nearly exactly the same as the classical solution obtained with finite differences. The error between the two solutions is of the order of 10^{-7}, which is 4 orders of magnitudes smaller than the error we asked for.

Once the validity of our solver has been checked on multiple test cases, the next interesting property we would like to verify is the asymptotic cost: does the implemented simulator seem to agree with the theoretical asymptotic complexities derived from [32] and [17]?

In our specific case, the Hamiltonian H to simulate can be decomposed in two 1-sparse Hermitian matrices, both of them having a spectral norm of 1. The exact decomposition can be found in appendix B 3. We chose to let the product-formula order be equal to $k = 1$ and reuse the asymptotic complexity found in eq. (15) by changing the time of simulation t by the time $f(t)$:

$$\mathcal{O}\left(5^{2k} n^2 f(t) \left(\frac{f(t)}{\epsilon}\right)^{\frac{1}{2k}}\right). \tag{17}$$
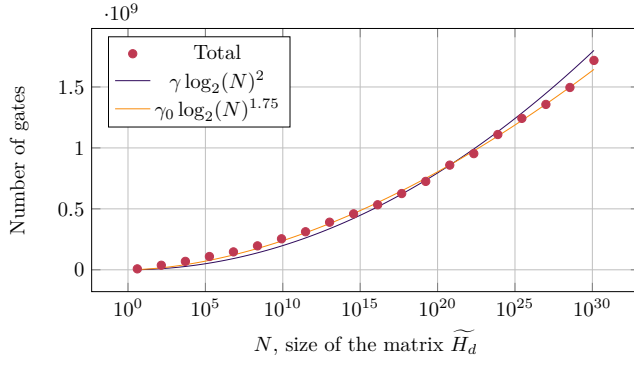
Following the study performed in [32],

$$f(t) = \frac{t}{\delta x} = t\left(N_d - 1\right) \tag{18}$$

where δx is the distance between two discretisation points. Moreover, it is possible to prove (see appendix B 3) that
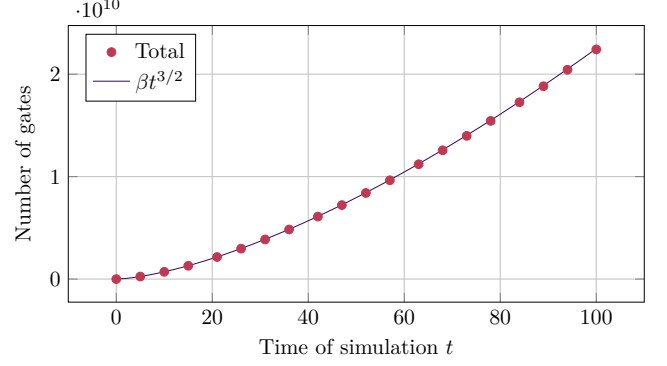
$$n = \lceil \log_2(2N_d - 1) \rceil \tag{19}$$

Replacing $f(t)$ and n in eq. (10) and eq. (11) gives us a gate complexity of
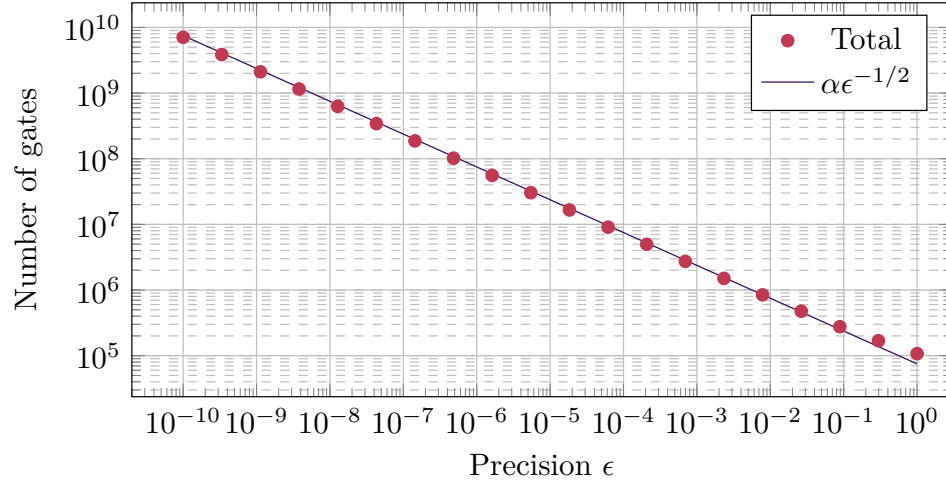
$$\mathcal{O}\left(5^k \log_2\left(N_d\right)^2\right) \tag{20}$$

(a)Number of quantum gates versus simulated matrix size. The values of the constants $\gamma = 250\,000$ and $\gamma_0 = 2\,000\,000$ have been chosen arbitrarily to fit the experimental data.

(b)Number of quantum gates versus physical time. The value of $\beta = 39\,000\,000$ has been chosen arbitrarily to fit the experimental data.

(c)Number of quantum gates versus targeted solution precision. The value of $\alpha = 130\,000$ has been chosen arbitrarily to fit the experimental data.

FIG. 1. Number of quantum gates needed to simulate the Hamiltonian described in appendix B using the oracles implemented following appendix C. Graphs generated with a Trotter-Suzuki product-formula order $k = 1$, 32 discretisation points (i.e. $n = 6$ qubits) for fig. 1(b) and fig. 1(c), a physical time $t = 1$ for fig. 1(a) and fig. 1(c) and a precision $\epsilon = 10^{-5}$ for fig. 1(a) and fig. 1(b).

to construct a circuit simulating $e^{-iHt/r}$ and a number of repetitions

$$r \in \mathcal{O}\left(5^k t N_d \left(\frac{t N_d}{\epsilon}\right)^{\frac{1}{2k}}\right). \tag{21}$$

Merging the two expression results in a gate complexity of

$$\mathcal{O}\left(5^{2k} t N_d \log_2(N_d)^2 \left(\frac{t N_d}{\epsilon}\right)^{\frac{1}{2k}}\right). \tag{22}$$

Choosing the Totter-Suzuki formula order $k = 1$ gives us a final complexity of

$$\mathcal{O}\left(N_d^{3/2} \log_2(N_d)^2 \frac{t^{3/2}}{\sqrt{\epsilon}}\right) \tag{23}$$

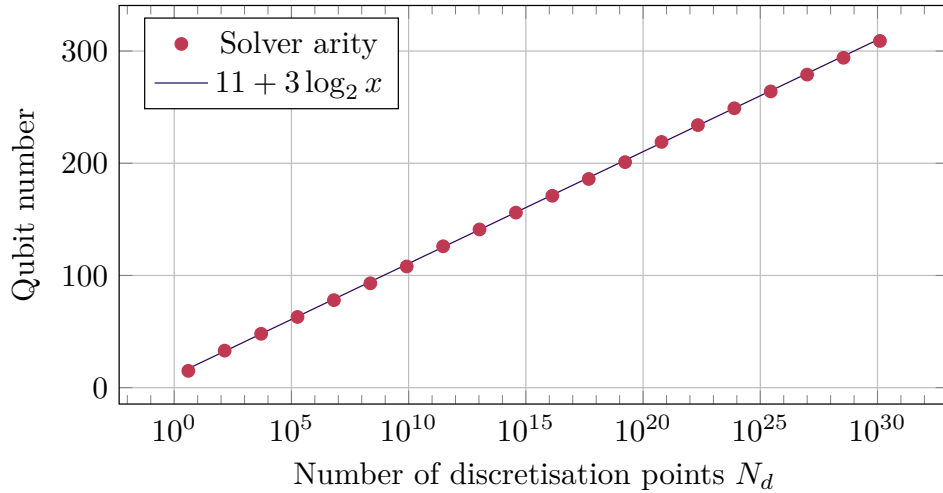to solve the wave equation presented in eq. (1). This theoretical result is verified experimentally in fig. 4(a).

FIG. 2. Plot of the number of logical qubits needed to run the wave equation solver for a time $t = 1$, a precision $\epsilon = 10^{-5}$ and a Trotter-Suzuki product-formula of order $k = 1$. The constants values 11 and 3 have been chosen arbitrarily to fit the experimental data. The number of physical qubits needed will depend on their error rate as noted in [36]. Multiplying the number of logical qubits by 3 to 4 orders of magnitude might be a good estimate of the actual number of physical qubits required.

(a)Quantum versus classical solution. Solutions are not visually distinguishable on the graph, see the associated absolute error.

(b)Absolute error between the solution obtained by a classical finite-difference solver and the solution computed with the quantum solver.
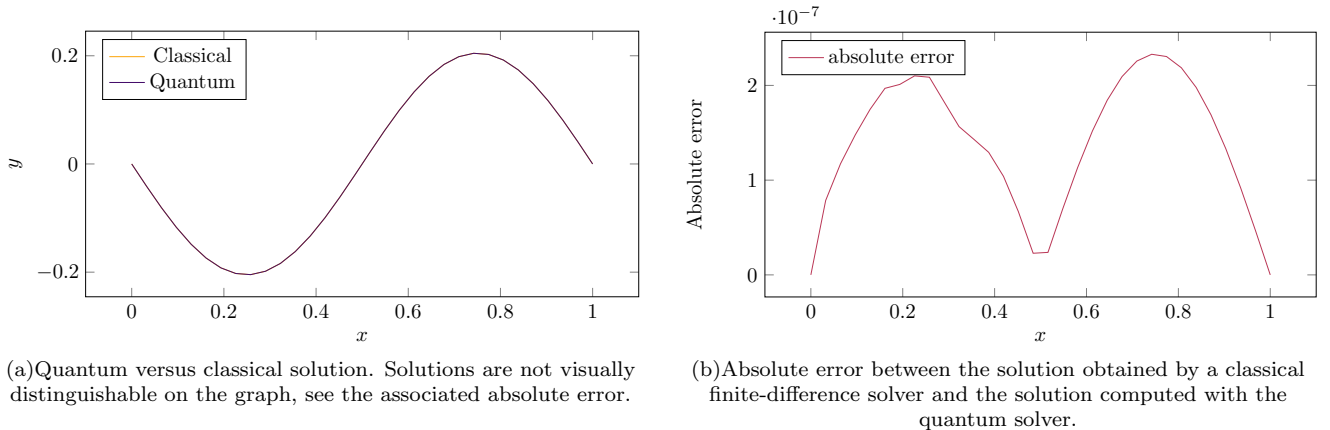
FIG. 3. Comparison of the classical solver and the quantum solver. Both solvers solved the 1-D wave equation with $N_d = 32$ discretisation points and a physical time of $t = 0.4$. The classical solver uses finite-differences with a very small time-step in order to avoid as much as possible errors due to time-discretisation. The quantum solver was instructed to solve the wave equation with a precision of at least $\epsilon = 10^{-3}$, used a Trotter-Suzuki order of $k = 1$. The solutions of the two solvers are too close to be able to notice a difference (they overlap on the graph), that is why a second graph plotting the absolute error between the two solvers is included.

V. DISCUSSION

In this work, we focus on the practical cost of implementing a 1-dimensional quantum wave equation solver on a quantum computer. We show that a quantum computer is able to solve partial differential equations by constructing and simulating the quantum circuits described. We also study the scaling of the solver with respect to several parameters of interest and show that the theoretical asymptotic bounds are mostly verified.

In future works, one can study the possibilities of circuit optimisation. It would also be interesting to implement Neumann boundary conditions instead of Dirichlet ones. A practical implementation including a non-constant propagation speed c has also been realised during the writing of this paper. The results were encouraging but were not judged mature enought to include them in the paper. Finally, future works might want to extend the wave equation solver to 2 dimensions or more.
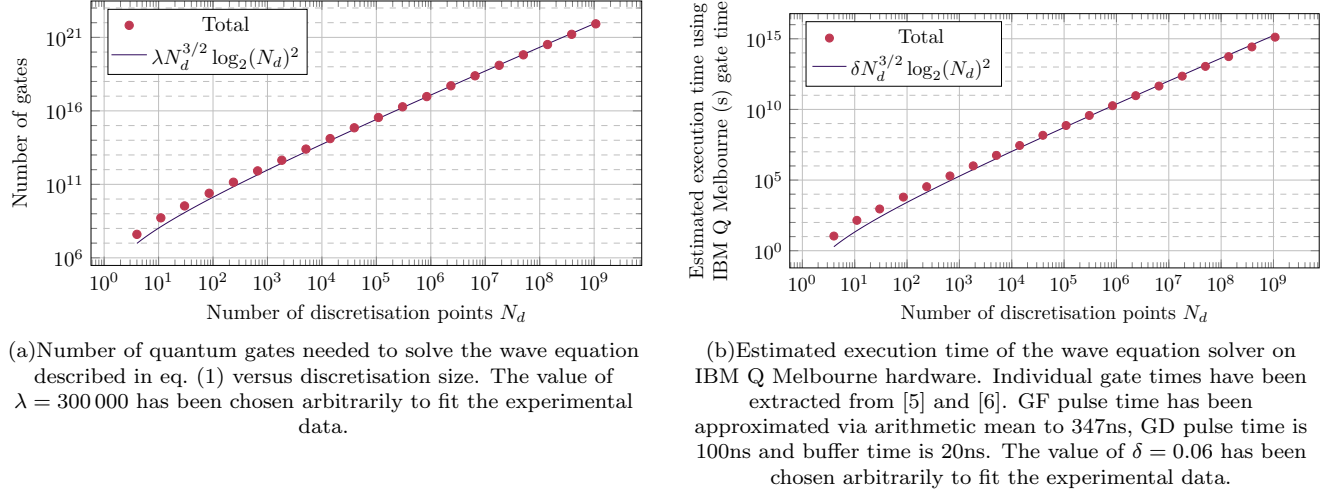
(a)Number of quantum gates needed to solve the wave equation described in eq. (1) versus discretisation size. The value of $\lambda = 300\,000$ has been chosen arbitrarily to fit the experimental data.

(b)Estimated execution time of the wave equation solver on IBM Q Melbourne hardware. Individual gate times have been extracted from [5] and [6]. GF pulse time has been approximated via arithmetic mean to 347ns, GD pulse time is 100ns and buffer time is 20ns. The value of $\delta = 0.06$ has been chosen arbitrarily to fit the experimental data.

FIG. 4. Graphs generated with a Trotter-Suzuki product-formula order $k = 1$, a physical time $t = 1$ and a precision $\epsilon = 10^{-5}$.

ACKNOWLEDGMENTS

SUPPLEMENTARY MATERIAL

The implementation of the quantum wave equation solver is available at `https://gitlab.com/cerfacs/qaths`. The `qprof` tool is available at `https://gitlab.com/qcomputing/qprof/qprof`.

[1] 2015. Constructing Large Controlled Nots. `https://algassert.com/circuits/2015/06/05/Constructing-Large-Controlled-Nots.html`. (2015). Accessed: 2020-03-27.

[2] 2019. 14-qubit backend: IBM Q team, "IBM Q 16 Melbourne backend specifications V1.3.0" (2019). (2019). Retrieved from `https://quantum-computing.ibm.com`.

[3] 2019. Hamiltonian simulation implementation in qiskit-aqua. `https://github.com/Qiskit/qiskit-aqua/blob/master/qiskit/aqua/operators/weighted_pauli_operator.py#L837`. (2019). Accessed: 2020-03-27.

[4] 2019. IBM Quantum Computing. `https://www.ibm.com/quantum-computing/`. (2019). Accessed: 2020-03-27.

[5] 2019. Melbourne gate specification. `https://github.com/Qiskit/ibmq-device-information/tree/master/backends/melbourne/V1#gate-specification`. (2019). Accessed: 2020-03-27.

[6] 2019. Melbourne hardware operation execution time. `https://github.com/Qiskit/ibmq-device-information/blob/master/backends/melbourne/V1/version_log.md#gate-specification`. (2019). Accessed: 2020-03-27.

[7] 2019. Quantum algorithms for the simulation of Hamiltonian dynamics. `https://github.com/njross/simcount`. (2019). Accessed: 2020-03-27.

[8] 2019. Quantum computing — Intel Newsroom. `https://newsroom.intel.com/press-kits/quantum-computing/`. (2019). Accessed: 2020-03-27.

[9] 2019. Quantum Supremacy Using a Programmable Superconducting Processor. `https://ai.googleblog.com/2019/10/quantum-supremacy-using-programmable.html`. (2019). Accessed: 2020-03-27.

[10] Graeme Robert Ahokas. 2004. *Improved Algorithms for Approximate Quantum Fourier Transforms and Sparse Hamiltonian Simulations.* Master's thesis. University of Calgary. `https://doi.org/10.11575/PRISM/22839`

[11] Juan Miguel Arrazola, Timjan Kalajdzievski, Christian Weedbrook, and Seth Lloyd. 2018. Quantum algorithm for non-homogeneous linear partial differential equations. (09 2018). arXiv:1809.02622v1 `http://arxiv.org/abs/1809.02622v1`

[12] Frank Arute, Kunal Arya, Ryan Babbush, Dave Bacon, Joseph C. Bardin, Rami Barends, Rupak Biswas, Sergio Boixo, Fernando G. S. L. Brandao, David A. Buell, Brian Burkett, Yu Chen, Zijun Chen, Ben Chiaro, Roberto Collins, William Courtney, Andrew Dunsworth, Edward Farhi, Brooks Foxen, Austin Fowler, Craig Gidney, Marissa Giustina, Rob Graff,

Keith Guerin, Steve Habegger, Matthew P. Harrigan, Michael J. Hartmann, Alan Ho, Markus Hoffmann, Trent Huang, Travis S. Humble, Sergei V. Isakov, Evan Jeffrey, Zhang Jiang, Dvir Kafri, Kostyantyn Kechedzhi, Julian Kelly, Paul V. Klimov, Sergey Knysh, Alexander Korotkov, Fedor Kostritsa, David Landhuis, Mike Lindmark, Erik Lucero, Dmitry Lyakh, Salvatore Mandrà, Jarrod R. McClean, Matthew McEwen, Anthony Megrant, Xiao Mi, Kristel Michielsen, Masoud Mohseni, Josh Mutus, Ofer Naaman, Matthew Neeley, Charles Neill, Murphy Yuezhen Niu, Eric Ostby, Andre Petukhov, John C. Platt, Chris Quintana, Eleanor G. Rieffel, Pedram Roushan, Nicholas C. Rubin, Daniel Sank, Kevin J. Satzinger, Vadim Smelyanskiy, Kevin J. Sung, Matthew D. Trevithick, Amit Vainsencher, Benjamin Villalonga, Theodore White, Z. Jamie Yao, Ping Yeh, Adam Zalcman, Hartmut Neven, and John M. Martinis. 2019. Quantum supremacy using a programmable superconducting processor. *Nature* 574 (10 2019), 505–510. Issue 7779. `https://doi.org/10.1038/s41586-019-1666-5`

[13] Alain Bamberger, Guy Chavent, and Patrick Lailly. 1977. Une application de la théorie du contrôle à un problème inverse de sismique. *Ann. Geophys* 33, 1 (1977), 2.

[14] Alain Bamberger, Guy Chavent, and Patrick Lailly. 1979. About the stability of the inverse problem in 1-D wave equations – application to the interpretation of seismic profiles. *Applied Mathematics & Optimization* 5 (3 1979), 1–47. Issue 1. `https://doi.org/10.1007/BF01442542`

[15] Adriano Barenco, Artur Ekert, Kalle-Antti Suominen, and Päivi Törmä. 1996. Approximate Quantum Fourier Transform and Decoherence. (Jan 1996). `https://doi.org/10.1103/PhysRevA.54.139` arXiv:quant-ph/9601018v1

[16] Dominic W. Berry. 2010. High-order quantum algorithm for solving linear differential equations. (10 2010). `https://doi.org/10.1088/1751-8113/47/10/105301` arXiv:1010.2745v2 J. Phys. A: Math. Theor. 47, 105301 (2014).

[17] Dominic W. Berry, Graeme Ahokas, Richard Cleve, and Barry C. Sanders. 2007. Efficient Quantum Algorithms for Simulating Sparse Hamiltonians. *Communications in Mathematical Physics* 270 (1 2007), 359–371. Issue 2. `https://doi.org/10.1007/s00220-006-0150-x` arXiv:quant-ph/0508139v2 Communications in Mathematical Physics 270, 359 (2007).

[18] Dominic W. Berry and Andrew M. Childs. 2012. Black-box Hamiltonian Simulation and Unitary Implementation. *Quantum Info. Comput.* 12, 1-2 (01 2012), 29–62. `https://doi.org/10.26421/QIC12.1-2` arXiv:0910.4157v4 Quantum Information and Computation 12, 29 (2012).

[19] Dominic W. Berry, Andrew M. Childs, Richard Cleve, Robin Kothari, and Rolando D. Somma. 2015. Simulating Hamiltonian Dynamics with a Truncated Taylor Series. *Physical Review Letters* 114 (3 2015). Issue 9. `https://doi.org/10.1103/PhysRevLett.114.090502` arXiv:1412.4687v1 Phys. Rev. Lett. 114, 090502 (2015).

[20] Dominic W. Berry, Andrew M. Childs, and Robin Kothari. 2015. Hamiltonian Simulation with Nearly Optimal Dependence on all Parameters. In *2015 IEEE 56th Annual Symposium on Foundations of Computer Science*. 792–809. `https://doi.org/10.1109/FOCS.2015.54` arXiv:1501.01715v3 Proceedings of the 56th IEEE Symposium on Foundations of Computer Science (FOCS 2015), pp. 792-809 (2015).

[21] Dominic W. Berry, Andrew M. Childs, Aaron Ostrander, and Guoming Wang. 2017. Quantum Algorithm for Linear Differential Equations with Exponentially Improved Dependence on Precision. *Communications in Mathematical Physics* 356 (12 2017), 1057–1081. Issue 3. `https://doi.org/10.1007/s00220-017-3002-y` arXiv:1701.03684v2 Communications in Mathematical Physics 356, 1057-1081 (2017).

[22] Carlos Bravo-Prieto, Ryan LaRose, M. Cerezo, Yigit Subasi, Lukasz Cincio, and Patrick J. Coles. 2019. Variational Quantum Linear Solver: A Hybrid Algorithm for Linear Systems. (09 2019). arXiv:1909.05820v1 `http://arxiv.org/abs/1909.05820v1`

[23] Yudong Cao, Jonathan Romero, Jonathan P. Olson, Matthias Degroote, Peter D. Johnson, Mária Kieferová, Ian D. Kivlichan, Tim Menke, Borja Peropadre, Nicolas P. D. Sawaya, Sukin Sim, Libor Veis, and Alán Aspuru-Guzik. 2018. Quantum Chemistry in the Age of Quantum Computing. (12 2018). arXiv:1812.09976v2 `http://arxiv.org/abs/1812.09976v2`

[24] Andrew M. Childs and Robin Kothari. 2011. Simulating Sparse Hamiltonians with Star Decompositions. In *Theory of Quantum Computation, Communication, and Cryptography*. Springer Berlin Heidelberg, 94–103. `https://doi.org/10.1007/978-3-642-18073-6_8` arXiv:1003.3683v2 Theory of Quantum Computation, Communication, and Cryptography (TQC 2010), Lecture Notes in Computer Science 6519, pp. 94-103 (2011).

[25] Andrew M. Childs and Jin-Peng Liu. 2019. Quantum spectral methods for differential equations. (01 2019). arXiv:1901.00961v1 `http://arxiv.org/abs/1901.00961v1`

[26] Andrew M. Childs, Jin-Peng Liu, and Aaron Ostrander. 2020. High-precision quantum algorithms for partial differential equations. (Feb 2020). arXiv:2002.07868v1 `http://arxiv.org/abs/2002.07868v1`

[27] Andrew M. Childs, Dmitri Maslov, Yunseong Nam, Neil J. Ross, and Yuan Su. 2018. Toward the first quantum simulation with quantum speedup. *Proceedings of the National Academy of Sciences* 115 (09 2018), 9456–9461. Issue 38. `https://doi.org/10.1073/pnas.1801723115` arXiv:1711.10980v1 Proceedings of the National Academy of Sciences 115, 9456-9461 (2018).

[28] Andrew M. Childs, Yuan Su, Minh C. Tran, Nathan Wiebe, and Shuchen Zhu. 2019. A Theory of Trotter Error. (Dec 2019). arXiv:1912.08854v1 `http://arxiv.org/abs/1912.08854v1`

[29] Andrew M. Childs and Nathan Wiebe. 2012. Hamiltonian Simulation Using Linear Combinations of Unitary Operations. (02 2012). `https://doi.org/10.26421/QIC12.11-12` arXiv:1202.5822v1 Quantum Information and Computation 12, 901-924 (2012).

[30] Richard Cleve and John Watrous. 2000. Fast parallel circuits for the quantum Fourier transform. (06 2000). arXiv:quant-ph/0006004v1 `http://arxiv.org/abs/quant-ph/0006004v1`

[31] Patrick J. Coles, Stephan Eidenbenz, Scott Pakin, Adetokunbo Adedoyin, John Ambrosiano, Petr Anisimov, William Casper, Gopinath Chennupati, Carleton Coffrin, Hristo Djidjev, David Gunter, Satish Karra, Nathan Lemons, Shizeng Lin, Andrey Lokhov, Alexander Malyzhenkov, David Mascarenas, Susan Mniszewski, Balu Nadiga, Dan O'Malley, Diane Oyen, Lakshman Prasad, Randy Roberts, Phil Romero, Nandakishore Santhi, Nikolai Sinitsyn, Pieter Swart, Marc Vuffray, Jim Wendelberger, Boram Yoon, Richard Zamora, and Wei Zhu. 2018. Quantum Algorithm Implementations for Beginners. (04 2018). arXiv:1804.03719v1 http://arxiv.org/abs/1804.03719v1

[32] Pedro C. S. Costa, Stephen Jordan, and Aaron Ostrander. 2019. Quantum algorithm for simulating the wave equation. *Physical Review A* 99 (1 2019). Issue 1. https://doi.org/10.1103/PhysRevA.99.012323 arXiv:1711.05394v1 Phys. Rev. A 99, 012323 (2019).

[33] Steven A. Cuccaro, Thomas G. Draper, Samuel A. Kutin, and David Petrie Moulton. 2004. A new quantum ripple-carry addition circuit. (10 2004). arXiv:quant-ph/0410184v1 http://arxiv.org/abs/quant-ph/0410184v1

[34] Thomas G. Draper. 2000. Addition on a Quantum Computer. (08 2000). arXiv:quant-ph/0008033v1 http://arxiv.org/abs/quant-ph/0008033v1

[35] Edward Farhi, Jeffrey Goldstone, and Sam Gutmann. 2014. A Quantum Approximate Optimization Algorithm. (Nov 2014). arXiv:1411.4028v1 http://arxiv.org/abs/1411.4028v1

[36] Austin G. Fowler, Matteo Mariantoni, John M. Martinis, and Andrew N. Cleland. 2012. Surface codes: Towards practical large-scale quantum computation. (2012). https://doi.org/10.1103/PhysRevA.86.032324 arXiv:quant-ph/1208.0928v2

[37] Juan José García-Ripoll. 2019. Quantum-inspired algorithms for multivariate analysis: from interpolation to partial differential equations. (09 2019). arXiv:1909.06619v1 http://arxiv.org/abs/1909.06619v1

[38] András Gilyén, Yuan Su, Guang Hao Low, and Nathan Wiebe. 2018. Quantum singular value transformation and beyond: exponential improvements for quantum matrix arithmetics. (06 2018). arXiv:1806.01838v1 http://arxiv.org/abs/1806.01838v1

[39] Thomas Häner, Martin Roetteler, and Krysta M. Svore. 2016. Factoring using 2n+2 qubits with Toffoli based modular multiplication. (11 2016). arXiv:1611.07995v2 http://arxiv.org/abs/1611.07995v2 Quantum Information and Computation, Vol. 17, No. 7 & 8 (2017).

[40] Aram W. Harrow, Avinatan Hassidim, and Seth Lloyd. 2009. Quantum Algorithm for Linear Systems of Equations. *Physical Review Letters* 103 (10 2009). Issue 15. https://doi.org/10.1103/PhysRevLett.103.150502 arXiv:0811.3171v3 Phys. Rev. Lett. vol. 15, no. 103, pp. 150502 (2009).

[41] Hsin-Yuan Huang, Kishor Bharti, and Patrick Rebentrost. 2019. Near-term quantum algorithms for linear systems of equations. (Sep 2019). arXiv:1909.07344v2 http://arxiv.org/abs/1909.07344v2

[42] Iordanis Kerenidis and Anupam Prakash. 2017. Quantum gradient descent for linear systems and least squares. (04 2017). arXiv:1704.04992v3 http://arxiv.org/abs/1704.04992v3

[43] Iordanis Kerenidis and Anupam Prakash. 2018. A Quantum Interior Point Method for LPs and SDPs. (08 2018). arXiv:1808.09266v1 http://arxiv.org/abs/1808.09266v1

[44] Maria Kieferova, Artur Scherer, and Dominic Berry. 2018. Simulating the dynamics of time-dependent Hamiltonians with a truncated Dyson series. (05 2018). arXiv:1805.00582v1 http://arxiv.org/abs/1805.00582v1 Only eprint on arXiv.

[45] Taewan Kim and Byung-Soo Choi. 2018. Efficient decomposition methods for controlled-Rnusing a single ancillary qubit. *Scientific Reports* 8, 1 (03 Apr 2018), 5445. https://doi.org/10.1038/s41598-018-23764-x

[46] Sarah K. Leyton and Tobias J. Osborne. 2008. A quantum algorithm to solve nonlinear differential equations. (12 2008). arXiv:0812.4423v1 http://arxiv.org/abs/0812.4423v1

[47] Guang Hao Low. 2018. Hamiltonian simulation with nearly optimal dependence on spectral norm. (07 2018). arXiv:1807.03967v1 http://arxiv.org/abs/1807.03967v1

[48] Guang Hao Low and Isaac L. Chuang. 2016. Hamiltonian Simulation by Qubitization. (10 2016). arXiv:1610.06546v2 http://arxiv.org/abs/1610.06546v2 Only available as eprint, no journal publication.

[49] Guang Hao Low and Isaac L. Chuang. 2017. Hamiltonian Simulation by Uniform Spectral Amplification. (07 2017). arXiv:1707.05391v1 http://arxiv.org/abs/1707.05391v1 Only available as eprint. No journal publication.

[50] Guang Hao Low and Isaac L. Chuang. 2017. Optimal Hamiltonian Simulation by Quantum Signal Processing. *Physical Review Letters* 118 (1 2017). Issue 1. https://doi.org/10.1103/PhysRevLett.118.010501 arXiv:1606.02685v2 Phys. Rev. Lett. 118, 010501 (2017).

[51] Michael Lubasch, Jaewoo Joo, Pierre Moinier, Martin Kiffner, and Dieter Jaksch. 2019. Variational Quantum Algorithms for Nonlinear Problems. (Jul 2019). arXiv:1907.09032v2 http://arxiv.org/abs/1907.09032v2

[52] Juan M. Pino, Joan M. Dreiling, Caroline. Figgatt, John P. Gaebler, Steven A. Moses, Charles H. Baldwin, Michael Foss-Feig, David Hayes, K. Mayer, Ciarán Ryan-Anderson, and Brian Neyenhuis. 2020. Demonstration of the QCCD trapped-ion quantum computer architecture. (Mar 2020). arXiv:2003.01293v2 http://arxiv.org/abs/2003.01293v1

[53] Neil J Ross and Peter Selinger. 2014. Optimal ancilla-free Clifford+ T approximation of z-rotations. *arXiv preprint arXiv:1403.2975* (2014).

[54] Artur Scherer, Benoît Valiron, Siun-Chuon Mau, Scott Alexander, Eric van den Berg, and Thomas E. Chapuran. 2017. Concrete resource analysis of the quantum linear-system algorithm used to compute the electromagnetic scattering cross section of a 2D target. *Quantum Information Processing* 16 (3 2017). Issue 3. https://doi.org/10.1007/s11128-016-1495-5 arXiv:1505.06552v2 Quantum Inf Process (2017) 16: 60.

[55] Changpeng Shao and Hua Xiang. 2020. Row and column iteration methods to solve linear systems on a quantum computer. *Phys. Rev. A* 101 (Feb 2020), 022322. Issue 2. https://doi.org/10.1103/PhysRevA.101.022322

[56] Vivek V. Shende and Igor L. Markov. 2008. On the CNOT-cost of TOFFOLI gates. (2008). arXiv:quant-ph/0803.2316

[57] David Shmoys, Dominic W. Berry, Andrew M. Childs, Richard Cleve, Robin Kothari, and Rolando D. Somma. 2014. Exponential improvement in precision for simulating sparse Hamiltonians. In *Proceedings of the 46th Annual ACM Symposium on Theory of Computing - STOC 14*. 283–292. `https://doi.org/10.1145/2591796.2591854` arXiv:1312.1414v2 Proceedings of the 46th ACM Symposium on Theory of Computing (STOC 2014), pp. 283-292 (2014).

[58] Siddhartha Srivastava and Veera Sundararaghavan. 2018. Box algorithm for the solution of differential equations on a quantum annealer. (12 2018). arXiv:1812.10572v2 `http://arxiv.org/abs/1812.10572v2`

[59] Masuo Suzuki. 1986. Quantum statistical monte carlo methods and applications to spin systems. *Journal of Statistical Physics* 43 (6 1986), 883–909. Issue 5-6. `https://doi.org/10.1007/BF02628318`

[60] Masuo Suzuki. 1990. Fractal decomposition of exponential operators with applications to many-body theories and Monte Carlo simulations. *Physics Letters A* 146 (6 1990), 319–323. Issue 6. `https://doi.org/10.1016/0375-9601(90)90962-N`

[61] Himanshu Thapliyal and Nagarajan Ranganathan. 2017. Design of Efficient Reversible Logic Based Binary and BCD Adder Circuits. (12 2017). `https://doi.org/10.1145/2491682` arXiv:1712.02630v1 J. Emerg. Technol. Comput. Syst. 9 (2013) 17:1-17:31.

[62] Blaga N. Todorova and René Steijl. 2020. Quantum algorithm for the collisionless Boltzmann equation. *J. Comput. Phys.* 409 (5 2020), 109347. `https://doi.org/10.1016/j.jcp.2020.109347`

[63] Almudena Carrera Vazquez. 2018. *Quantum Algorithm for Solving Tri-Diagonal Linear Systems of Equations*. Master's thesis. ETH Zürich.

[64] Vlatko Vedral, Adriano Barenco, and Artur Ekert. 1996. Quantum networks for elementary arithmetic operations. *Physical Review A* 54, 1 (Jul 1996), 147–153. `https://doi.org/10.1103/physreva.54.147`

[65] Tao Xin, Shijie Wei, Jianlian Cui, Junxiang Xiao, Iñigo Arrazola, Lucas Lamata, Xiangyu Kong, Dawei Lu, Enrique Solano, and Guilu Long. 2018. A Quantum Algorithm for Solving Linear Differential Equations: Theory and Experiment. (07 2018). arXiv:1807.04553v1 `http://arxiv.org/abs/1807.04553v1`

[66] Xiaosi Xu, Jinzhao Sun, Suguru Endo, Ying Li, Simon C. Benjamin, and Xiao Yuan. 2019. Variational algorithms for linear algebra. (09 2019). arXiv:1909.03898v1 `http://arxiv.org/abs/1909.03898v1`

Appendix A: Product-formula implementation details

1. Hamiltonian simulation

Hamiltonian simulation is the problem of constructing a quantum circuit that will evolve a quantum state according to a Hamiltonian matrix, following the Schrödinger equation. In other words, Hamiltonian simulation algorithms generate a quantum circuit performing the unitary transformation U such that $||U - e^{-iHt}|| < \epsilon$, H being a given Hamiltonian matrix, t a time of evolution and ϵ a precision with respect to $||\cdot||$, the spectral norm.

Several quantum algorithms have been developed in the last few years to solve the problem of s-sparse Hamiltonian simulation [17–20, 24, 29, 44, 47–50, 57]. Among these algorithms we decided to implement the product-formula approach [10, 17], for the reasons presented in section III A.

The product formula algorithm has three main steps: decompose, simulate, recompose. It works by first decomposing the s-sparse Hamiltonian matrix H that should be simulated as a sum of Hermitian matrices H_j that are considered easy to simulate

$$H = \sum_{j=0}^{m-1} H_j. \tag{A1}$$

The second step is then to simulate each H_j separately, i.e. to create quantum circuits implementing $e^{-iH_j t}$ for all the H_j in the decomposition in eq. (A1). The last step uses the simulations computed in step two to approximate e^{-iHt}.

The very first questions that should be answered before starting any implementation of the product-formula algorithm are "What is an easy to simulate matrix?" and "What kind of Hermitian matrices are easy to simulate?".

2. Easy to simulate matrices

One of the most desirable properties for an "easy to simulate" matrix is the possibility to simulate it exactly, i.e. to construct a quantum circuit that will perfectly implement e^{-iHt}. This property becomes a requirement when one wants rigorous bounds on the error of the final simulation. Another enviable property of these matrices is that they can be simulated with a low gate number and only a few calls to the matrix oracle.

Definition 3 (Easy to simulate matrix). A Hermitian matrix H can be qualified as "easy to simulate" if there exist an algorithm that takes as input a time t and the matrix H and outputs a quantum circuit $C(H)_t$ such that

1. The quantum circuit $C(H)_t$ implements exactly the unitary transformation e^{-iHt}, i.e.

$$\left|\left|e^{-iHt} - C(H)_t\right|\right| = 0.$$

2. The algorithm only needs $\mathcal{O}(1)$ calls to the oracle of H and $\mathcal{O}(\log N)$ additional gates, N being the dimension of the matrix H.

With this definition of an "easy to simulate" matrix, we can now search for matrices or group of matrices that satisfy this definition.

a. Multiples of the identity

The first and easiest matrices that fulfil the easy to simulate matrix requirements are the multiples of the identity matrix $\{\alpha I,\ \alpha \in \mathbb{R}\}$ with I the identity matrix. The quantum circuit to simulate this class of matrices can be found in [63].

b. 1-sparse Hermitian matrices

A larger class of matrices that can be efficiently and exactly simulated are the 1-sparse, integer weighted, Hermitian matrices. Quantum circuits simulating exactly 1-sparse matrices with integer weights can be found in [10].

Note 3. Procedures simulating 1-sparse matrices with real (non-integers) weights are also described in the paper, but these matrices do not fall in the "easy to simulate" category because the procedures explained are exact only if all the matrix weights can be represented exactly with a fixed-point representation, which is not always verified.

Note 4. Multiples of identity matrices presented in appendix A 2 a are a special case of 1-sparse matrices. The two classes have been separated because more efficient quantum circuits exists for αI matrices.

3. Decomposition of H

Once the set of "easy to simulate" matrices has been established, the next step of the algorithm is to decompose the s-sparse matrix H as a sum of matrices in this set.

There are two possible ways of performing this decomposition, each one with its advantages and drawbacks: applying a procedure computing the decomposition automatically, or decompose the matrix H beforehand and provide the decomposition to the algorithm.

The first solution, which is to automatically construct the oracles of the H_j matrices from the oracle of the H matrix has been studied in [10] and [24]. Thanks to this automatic decomposition procedure, we only need to implement one oracle. This simplicity comes at the cost of a higher gate count: each call to the automatically constructed oracles of the matrices H_j will require several calls to the oracle of H along with additional gates.

On the other hand, the second solution offers more control at the cost of less abstraction and more work. The decomposition of H is not automatically computed and should be performed beforehand. Once the matrix H has been decomposed as in eq. (9), the oracles for the matrices H_j should be implemented. This means that we should now implement m oracles instead of only 1 for the first solution. The main advantage of this method over the one using automatic-decomposition is that it gives us more control, a control that can be used to optimize even more the decomposition of eq. (A1) (less H_j in the decomposition, H_j matrices that can be simulated more efficiently, ...).

All the advantages and drawbacks weighted, we chose to implement the second option for several reasons. First, the implementation of the automatic decomposition procedure adds a non-negligible implementation complexity to the whole Hamiltonian simulation procedure. Moreover, the automatic decomposition procedure can be implemented afterwards and plugged effortlessly to the non-automatic implementation. Finally, our use-case only required to simulate a 2-sparse Hamiltonian that can be decomposed as the sum of two 1-sparse, easy to simulate, Hermitian matrices, which makes the manual decomposition step manageable.

4. Simulation of the H_j

Once the matrix H has been decomposed following eq. (A1) with each H_j being an "easy to simulate" matrix, the simulation of H_j becomes a straightforward application of the procedures described in appendix A 2.

After this step, we have access to quantum circuits implementing $e^{-iH_j t}$ for $j \in [0, m-1]$ and $t \in \mathbb{R}$.
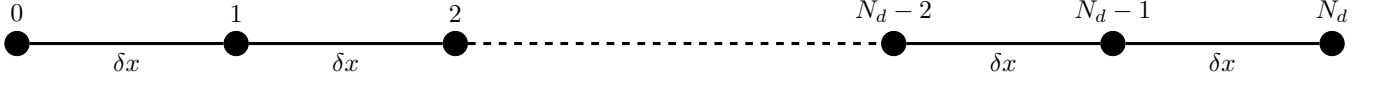
FIG. 5. Graph $G_{\delta x}$ built from the discretisation of the 1-dimensional line $[0, 1]$ with N_d discretisation points (i.e. $\delta x = \frac{1}{N_d-1}$).

5. Re-composition of the $e^{-iH_j t}$

The ultimate step of the algorithm is to approximate the desired evolution e^{-iHt} with the evolutions $e^{-iH_j t}$. In the special case of mutually commuting H_j, this step is trivial as it boils down to use the properties of the exponential function on matrices and write $e^{iHt} = e^{i\sum_j H_j t} = \prod_j e^{iH_j t}$. But in the more realistic case where the matrices H_j do not commute, a more sophisticated method should be used to approximate the evolution e^{-iHt}. To this end, we used the first-order Lie-Trotter-Suzuki product formula defined in Definition definition 4.

Definition 4 (Lie-Trotter-Suzuki product formula [27, 59, 60]). The Lie-Trotter-Suzuki product formula approximates

$$\exp\left(\lambda \sum_{j=0}^{m-1} \alpha_j H_j\right) \tag{A2}$$

with

$$S_2(\lambda) = \prod_{j=0}^{m-1} e^{\alpha_j H_j \lambda/2} \prod_{j=m-1}^{0} e^{\alpha_j H_j \lambda/2} \tag{A3}$$

and can be generalized recursively to higher-orders

$$S_{2k}(\lambda) = [S_{2k-2}(p_k\lambda)]^2 \times S_{2k-1}((1-4p_k)\lambda)[S_{2k-2}(p_k\lambda)]^2 \tag{A4}$$

with $p_k = \left(4 - 4^{1/(2k-1)}\right)^{-1}$ for $k > 1$. Using this formula, we have the approximation

$$e^{\lambda H} = \left[S_{2k}\left(\frac{\lambda}{n}\right)\right]^n + \mathcal{O}\left(\frac{|\lambda|^{2k+1}}{n^{2k}}\right). \tag{A5}$$

We used the Lie-Trotter-Suzuki product formula with $\lambda = -it$ to approximate the operator e^{-iHt} up to an error of $\epsilon \in \mathcal{O}\left(\frac{t^{2k+1}}{n^{2k}}\right)$.

Appendix B: Hermitian matrix construction and decomposition

One of the main challenge in implementing a quantum wave equation solver lies in the construction and implementation of the needed oracles. This appendix describes the first step of the implementation process: the construction and decomposition of the Hamiltonian matrix that will be simulated using the Hamiltonian simulation procedure introduced in appendix A.

This appendix follows the analysis performed in [32] and adds details and observations that will be refereed to in appendix C when dealing with the actual oracle implementation.

1. Hamiltonian matrix description

In order to devise the Hamiltonian matrix that should be simulated to solve the wave equation, the first step is to discretise eq. (1) with respect to space. Such a discretisation can be seen as a graph $G_{\delta x}$ whose vertices are the discretisation points and with edges between nearest neighbour vertices. The graph $G_{\delta x}$ is depicted in fig. 5.

The graph Laplacian of $G_{\delta x}$, defined as

$$L(G_{\delta x})_{i,j} := \begin{cases} \deg(v_i) & \text{if } i = j \\ -1 & \text{if } (i \neq j) \wedge (v_i \text{ adjacent to } v_j) \\ 0 & \text{otherwise} \end{cases} \tag{B1}$$

can then be used to approximate the differential operator $\frac{\partial^2}{\partial x^2}$. By using the discretisation approximation

$$\frac{\partial^2 \phi}{\partial x^2}(i\delta x, t) \approx \frac{\phi_{i-1,t} - 2\phi_{i,t} + \phi_{i+1,t}}{\delta x^2} \tag{B2}$$

with $\phi_{i,t} = \phi(i\delta x, t)$, and approximating $\phi(x, t)$ with a vector $\phi = [\phi_{i,t}]_{0 \leqslant i < N_d}$, the matrix

$$A = -\frac{1}{\delta x^2} L(G_{\delta x}) \tag{B3}$$

approximates the second derivative of ϕ when $\delta x \to 0$ as

$$[A\phi]_i = \frac{\phi((i-1)\delta x, t) - 2\phi(i\delta x, t) + \phi((i+1)\delta x, t)}{\delta x^2}. \tag{B4}$$

The approximation in eq. (B3) is then used in eq. (1) to approximate the spatial derivative operator:

$$\frac{\partial^2}{\partial t^2}\phi = -\frac{1}{\delta x^2} L(G_{\delta x})\phi. \tag{B5}$$

Based on this formula, [32] shows that simulating

$$H = \begin{pmatrix} 0 & B \\ B^\dagger & 0 \end{pmatrix} \tag{B6}$$

with

$$BB^\dagger = L(G_{\delta x}) \tag{B7}$$

constructs a quantum circuit that will evolve a part of the quantum state it is applied on according to the discretised wave equation in eq. (B5).

A matrix B satisfying eq. (B7) can be obtained directly from the graph $G_{\delta x}$ representing the discretisation. The algorithm to construct the matrix B can be decomposed in three steps. First, the vertices (discretisation points) should be arbitrarily ordered by assigning them a unique index in $[0, N_d - 1]$. Then, each edge of the graph is arbitrarily oriented and indexed with indices in $[0, N_d - 2]$. Finally, B is computed with the following definition

$$B_{ij} = \begin{cases} 1 & \text{if edge } j \text{ is a self-loop of vertex } i, \\ 1 & \text{if edge } j \text{ has vertex } i \text{ as source}, \\ -1 & \text{if edge } j \text{ has vertex } i \text{ as sink}, \\ 0 & \text{otherwise} \end{cases} \tag{B8}$$

Note that edges' orientation and vertices/edges ordering is completely arbitrary. Changing either the edges orientation on one of the orderings will change the matrix B but will not affect $BB^\dagger$ which should be equal to $L(G_{\delta x})$. This freedom in the ordering and orientation choices takes a crucial importance in the oracle implementation as it allows us to pick the ordering/orientation that will produce an easy-to-implement matrix B.

2. Dirichlet boundary conditions

Fixing boundary conditions is a requirement for most of the partial differential equations to admit a unique well-defined solution. There exist several boundary conditions such as Neumann, Dirichlet, Robin or Cauchy ones. For simplicity, we restricted ourselves to the study of eq. (1) with Dirichlet boundary condition of eq. (2).

In the case of Dirichlet boundary conditions on the 1-dimensional line $[0, 1]$, the two boundary nodes at $x = 0$ and $x = 1$ can be ignored as their value is always equal to 0. Moreover, [32] shows that the graph $G_{\delta x}^D$ representing the discretisation with Dirichlet boundary conditions of eq. (2) is simply $G_{\delta x}$ with self-loops on the two outer nodes (i.e. the ones indexed 1 and $N_d - 2$ as 0 and $N_d - 1$ are ignored). $G_{\delta x}^D$ is depicted in fig. 6. The algorithm to construct the matrix B remain the same as explained in appendix B 1.
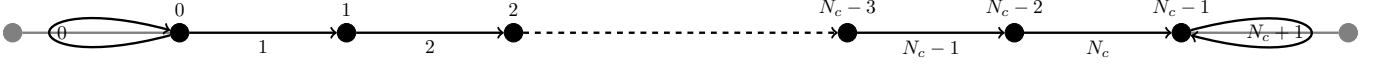
FIG. 6. Graph $G_{\delta x}^D$ representing the discretisation of the 1-dimensional line $[0, 1]$ with Dirichlet boundary conditions. The points and edges in grey are only drawn for illustration purpose and are ignored in the analysis because the boundary condition impose a value of 0 on these vertices. Loops are added to $G_{\delta x}$ to encode the fact that this graph represents Dirichlet boundary conditions. Vertices (resp. edges) are ordered with indices within $[0, N_c - 1]$ (resp. $[0, N_c + 1]$). N_c is the number of considered points and is equal to $N_d - 2$ (the two extremal points are ignored).

3. Matrices construction

All the pieces are now in place to start building the matrix $B_d \in \mathbb{R}^{(N_c-1) \times N_c}$. Using the definition of the matrix B written in eq. (B8) and the graph $G_{\delta x}^D$ depicted in fig. 6 we end up with

$$B_d = \begin{pmatrix} 1 & 1 & 0 & \cdots & 0 \\ 0 & -1 & 1 & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & -1 & 1 \end{pmatrix} \tag{B9}$$

We can easily check that $B_d B_d^\dagger$ is equal to the well-known discretisation matrix

$$B_d B_d^\dagger = \begin{pmatrix} 2 & -1 & 0 & \cdots & 0 \\ -1 & 2 & \ddots & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & 2 & -1 \\ 0 & \cdots & 0 & -1 & 2 \end{pmatrix}, \tag{B10}$$

which validate the method of construction of B_d.

Computing $\widetilde{H_d}$, the Hamiltonian matrix that should be simulated to evolve the quantum state according to the wave equation in eq. (1) with Dirichlet boundary conditions, is now straightforward. Using eq. (B6), we directly obtain

$$\widetilde{H_d} = \frac{1}{\delta x} \begin{pmatrix} 0 & \cdots & \cdots & 0 & 1 & 1 & 0 & \cdots & 0 \\ \vdots & & & \vdots & 0 & -1 & 1 & \ddots & \vdots \\ \vdots & & & \vdots & \vdots & \ddots & \ddots & \ddots & 0 \\ 0 & \cdots & \cdots & 0 & 0 & \cdots & 0 & -1 & 1 \\ 1 & 0 & \cdots & 0 & 0 & \cdots & \cdots & \cdots & 0 \\ 1 & -1 & \ddots & \vdots & \vdots & & & & \vdots \\ 0 & 1 & \ddots & 0 & \vdots & & & & \vdots \\ \vdots & \ddots & \ddots & -1 & \vdots & & & & \vdots \\ 0 & \cdots & 0 & 1 & 0 & \cdots & \cdots & \cdots & 0 \end{pmatrix} \tag{B11}$$

As explained in appendix A, the Hamiltonian simulation algorithm implemented requires that the Hamiltonian to simulate is split as a sum of 1-sparse Hermitian matrices. There are a lot of valid decompositions for the matrix $\widetilde{H_d}$ and we are free to choose the decomposition that will simplify the most the oracle implementation or reduce the gate complexity.

We made the choice to decompose B_d as two 1-sparse matrices and then reflect this decomposition on $\widetilde{H_d}$. Let B_1

and B_{-1} defined as

$$B_1 = \begin{pmatrix} 0 & 1 & 0 & \cdots & 0 \\ \vdots & \ddots & 1 & \ddots & \vdots \\ \vdots & & \ddots & \ddots & 0 \\ 0 & \cdots & \cdots & 0 & 1 \end{pmatrix} \tag{B12}$$

$$B_{-1} = \begin{pmatrix} 1 & 0 & \cdots & \cdots & 0 \\ 0 & -1 & \ddots & & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & -1 & 0 \end{pmatrix} \tag{B13}$$

we have $B_{\mathrm{d}} = B_1 + B_{-1}$. Let also

$$\widetilde{H_1} = \frac{1}{\delta x} \begin{pmatrix} 0 & B_1 \\ B_1^\dagger & 0 \end{pmatrix}, \quad \widetilde{H_{-1}} = \frac{1}{\delta x} \begin{pmatrix} 0 & B_{-1} \\ B_{-1}^\dagger & 0 \end{pmatrix}, \tag{B14}$$

it is easy to see that $\widetilde{H_d} = \widetilde{H_1} + \widetilde{H_{-1}}$ and that both $\widetilde{H_1}$ and $\widetilde{H_{-1}}$ are 1-sparse Hermitian matrices.

For convenience, we also define

$$H_1 = \begin{pmatrix} 0 & B_1 \\ B_1^\dagger & 0 \end{pmatrix}, \quad H_{-1} = \begin{pmatrix} 0 & B_{-1} \\ B_{-1}^\dagger & 0 \end{pmatrix}, \tag{B15}$$

and $H_d = H_1 + H_{-1}$, the $\widetilde{H_1}$, $\widetilde{H_{-1}}$ and $\widetilde{H_d}$ matrices rescaled to contain only integer weights. These matrices have the interesting property that simulating $\widetilde{H_d}$ (resp. $\widetilde{H_1}$, $\widetilde{H_{-1}}$) for a time t is equivalent to simulating H_d (resp. H_1, H_{-1}) for a time $\frac{t}{\delta x}$. This property will be used in the following sections as it offers us the opportunity to simulate the integer-weighted matrices H_d, H_1 and H_{-1} instead of the real-weighted ones $\widetilde{H_d}$, $\widetilde{H_1}$ and $\widetilde{H_{-1}}$.

Note also that a lower bound of the number of qubits needed to solve the wave equation for N_d discretisation points can be computed from the dimensions of H_d. As the non-empty upper-left block of matrix H_d is of dimension $(2N_d - 1) \times (2N_d - 1)$, we need at least

$$\lceil \log_2(2N_d - 1) \rceil \tag{B16}$$

qubits to simulate it. This estimation does not take into account ancilla qubits that may be needed to implement the oracles.

Appendix C: Oracle construction

Oracles can be seen as the interface between a quantum procedure and real-world data. Their purpose is to encode classical data such that a quantum algorithm can process it efficiently.

1. Oracle interface

In order to work as a bridge between the classical and the quantum worlds and to be used by the quantum algorithm, a clear interface for the oracle should be established.

We chose to use the interface described in [10, Eq. 4.4] with slight modifications improving the arity of the oracle for our specific case of 1-sparse matrices.

More precisely, our oracles O implement the following interface

$$O \left| x_0 \right\rangle_x \left| 0 \right\rangle_m \left| 0 \right\rangle_v \left| 0 \right\rangle_s = \left| x_0 \right\rangle_x \left| m(x_0) \right\rangle_m \left| v(x_0) \right\rangle_v \left| s(x_0) \right\rangle_s \tag{C1}$$

with $\left| x_0 \right\rangle_x$ encoding a row index as a unsigned integer, $m(x)$ the function that returns the column index of the only non-zero element in row x, $v(x) = |w(x)|$ the absolute value of the weight $w(x)$ of the first (and only) non-zero element in row x and

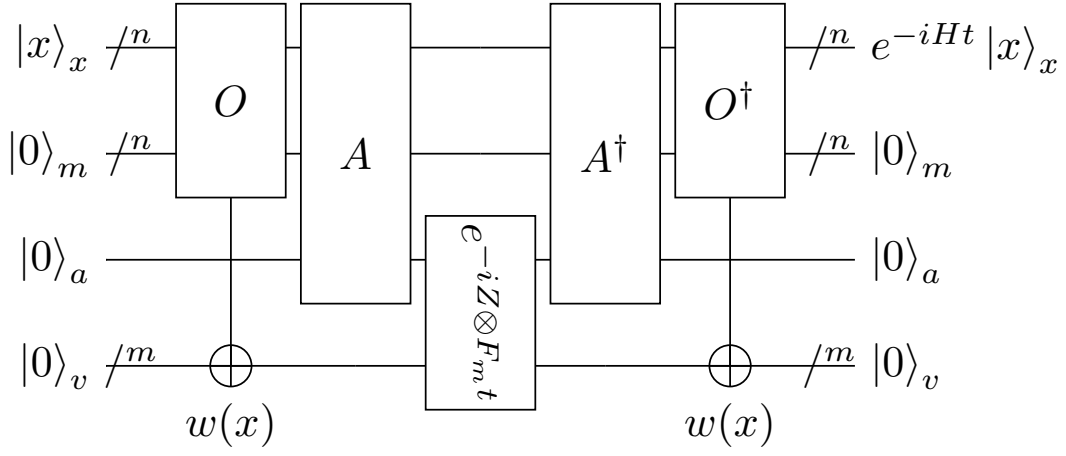$$s(x) = \begin{cases} 0 \text{ if } w(x) \geqslant 0 \\ 1 \text{ else} \end{cases} \tag{C2}$$

FIG. 7. Quantum circuit re-created from [10, p. 71] that simulates a 1-sparse integer-weighted Hamiltonian for a given time t. O is the implementation of the oracle, A is a quantum circuit defined in [10, p. 70]. F_m is defined as the diagonal matrix with diagonal entries increasing from 0 to $2^m - 1$ (see eq. (C7)).

the sign of the first non-zero entry in row x.

Note that the quantum register are labeled with their respective usage: x for the index of of the row considered, m for the index of the column considered, v for the value of the element at (row, index) and s for the sign of the element at (row, index). A fifth label "a" is used along the paper to label a register used as an ancilla.

The interface of the oracle O can also be obtained with 3 separate oracles that will each take care of computing one output:

$$M |x_0\rangle_x |0\rangle_m = |x_0\rangle_x |m(x)\rangle_m \tag{C3}$$

$$V |x_0\rangle_x |0\rangle_v = |x_0\rangle_x |v(x)\rangle_v \tag{C4}$$

$$S |x_0\rangle_x |0\rangle_s = |x_0\rangle_x |s(x)\rangle_s \tag{C5}$$

2. Optimisation of M and S

Claim 1. *The simulation algorithms provided by [10] have the interesting property that if the oracle V encodes a weight of zero for some inputs (i.e. $v(x) = 0$ for some x) then the outputs of oracles M and S are ignored for those inputs.*

Proof. The circuit simulating a 1-sparse m-bit-integer weighted Hamiltonian H depicted in fig. 7 is taken from [10].

In our special case of 1-bit weights (i.e. $m = 1$), the third quantum gate $e^{-iZ \otimes F_m t}$ can be written as

$$
\begin{aligned}
e^{-iZ \otimes F_m t} &= e^{-iZ \otimes F_1 t} \\
&= \exp\left[-i \begin{pmatrix} F_1 & 0 \\ 0 & -F_1 \end{pmatrix} t\right] \\
&= \begin{pmatrix} e^{-iF_1 t} & 0 \\ 0 & e^{iF_1 t} \end{pmatrix} \\
&= \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & e^{-it} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & e^{-it} \end{pmatrix}.
\end{aligned}
\tag{C6}
$$

where

$$F_m = \begin{pmatrix} 0 & 0 & \cdots & \cdots & \cdots & \cdots & & 0 \\ 0 & 1 & \ddots & & & & & \vdots \\ \vdots & \ddots & 2 & \ddots & & & & \vdots \\ \vdots & & \ddots & 3 & \ddots & & & \vdots \\ \vdots & & & \ddots & 4 & \ddots & & \vdots \\ \vdots & & & & \ddots & \ddots & 0 & \\ 0 & \cdots & \cdots & \cdots & \cdots & 0 & 2^m - 1 \end{pmatrix}. \tag{C7}$$

It follows from the matrix notation that if the second qubit $e^{-iZ \otimes F_1 t}$ is applied on is in the state $|0\rangle$, the gate $e^{-iZ \otimes F_1 t}$ is the identity transformation, i.e. the unitary operation $e^{-iZ \otimes F_1 t}$ sends $|00\rangle$ (resp. $|10\rangle$) to $|00\rangle$ (resp. $|10\rangle$). This means that if the oracle O does not set the last qubit to $|1\rangle$ (i.e. the oracle encodes a weight of 0 for the xth row of H), the quantum circuit depicted in fig. 7 can be simplified up to an identity transformation as the effects of O (resp. A) are reverted by $O^\dagger$ (resp. $A^\dagger$).

Rephrasing, if the xth row of matrix H has no non-zero entries, the effects of the oracle O is ignored, which implies that the effects of the oracles M and S that compose O are also ignored. $\square$

Using the result of Claim claim 1, we are free to implement any transformation that best suits us for the set of inputs $|x\rangle$ such that the xth row of the considered Hermitian matrix (H_1 or H_{-1}) has no non-zero elements as long as the oracle V implements the right transformation.

To illustrate clearly the *implemented* transformations we chose to encode with M and S, the next sections will re-write the matrices H_1 and H_{-1} according to eq. (B15) but with one **0** or $-\mathbf{0}$ in each empty row. A **0** entry at position (i, j) in the matrix means that the row i was empty, the oracle M will map $|i\rangle_x$ to $|j\rangle_m$ and the oracle S will encode a positive sign, i.e. $|0\rangle_s$. The same reasoning applies for $-\mathbf{0}$ entries, except that the encoded sign is now negative, i.e. $|1\rangle_s$.

The following sections will explain step by step the construction of each of the three oracles M, V and S, both for the matrix H_1 (M_1, V_1 and S_1) and the matrix H_{-1} (M_{-1}, V_{-1} and S_{-1}).

3. About arithmetic and logic quantum gates

Implementing the oracles M, V and S for the matrices H_1 and H_{-1} requires several arithmetic and logic quantum gates such as `or`, `add` or `compare`.

All these gates have been implemented prior to the oracle implementation and the implementation steps are detailed in this section.

a. The `or` gate

The `or` gate is easily implemented using only `X` and `CCX` (or Toffoli) gates. The implementation used is depicted in fig. 8 and uses the famous Boole algebra formula linking `not`, `or` and `and`: $x \vee y = \neg(\neg x \wedge \neg y)$.

b. The `add` and `sub` gates

Most of the research papers presenting an implementation of the `add` or `sub` gates only consider the case where the two numbers to add or subtract are stored in quantum registers.

In our case, the oracles implementation requires an adder and subtractor that can add or subtract to a quantum register a quantity known when the quantum circuit is generated, i.e. not necessarily encoded on a quantum state.

Claim 2. *Implementing a subtractor is trivial once an adder procedure is available.*

Proof. A subtractor can be implemented from a generic adder by using the identity

$$a - b = (a' + b)' \tag{C8}$$
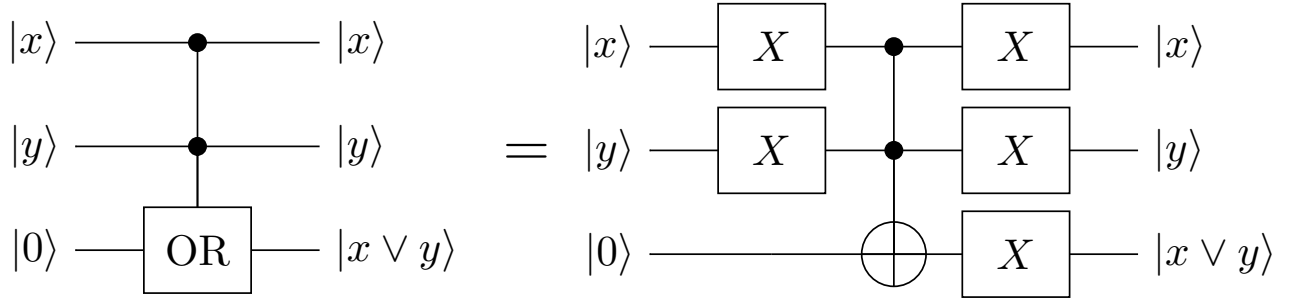
FIG. 8. Implementation of the `or` gate.
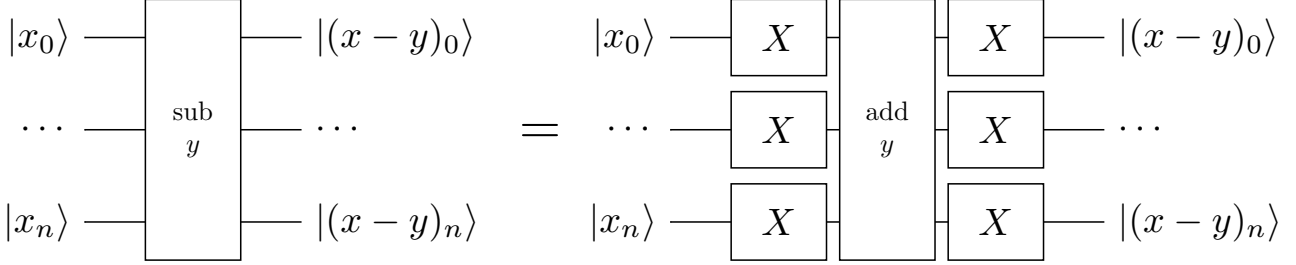
FIG. 9. Implementation of the `sub` gate from an `add` gate. The y value encoding is intentionally omitted. The subtractor will use the same encoding as the adder (i.e. either the y value is encoded on a quantum register or it is encoded directly in the quantum circuit implementing the adder). Note that the y value is not negated.

where $'$ denotes the bitwise complementation.

The circuit resulting of the application of this identity is depicted in fig. 9 and only requires one call to the adder and $2n$ additional gates, n being the number of qubits used to represent one of the operands.

$\square$

Note 5. Following Claim claim 2 we will restrict the study to implementing an adder. Implementing a subtractor is trivial and cheap in term of additional quantum gates used once an adder is available.

Definition 5. *Generation-time value* A generation-time value is a value that is known by the programmer when generating the quantum circuit. Knowing a value at generation-time may allow to optimise even further the generated quantum circuit. The closest analog in classical programming would be C-like macros or recent C++ constexpr expressions.

The easiest solution to overcome the problem caused by the non-compatible input formats between our problem (with a generation-time value) and the existing adders (with two values encoded on quantum registers) is to encode the quantity known at generation-time into ancillary qubits and then use the regular adder algorithms to add to a quantum register the value encoded in a second quantum register. Even if this solution is trivial to implement, it has the huge downside of requiring $\mathcal{O}\left(\log_2 b\right)$ additional ancillary qubits to temporarily store the generation-time value b.

Another answer to the problem would be to adapt a quantum adder originally devised to add two quantum registers to a quantum adder capable of adding a constant value to a quantum register. Several adders [33, 34, 61] have been studied to check if they can be modified to allow a generation-time input, i.e. if it possible to remove completely the quantum register storing the right-hand-side (or left-hand-side) of the addition.

The task of removing the quantum register storing one of the operands appears to be challenging for adders based on classical arithmetic like [33, 61] but trivial for Draper's quantum adder introduced in [34].

Claim 3. *Draper's quantum adder can be adapted into an efficient adder that takes as right-hand side input a unsigned "generation-time" integer value and add this value to a sufficiently large quantum register encoding another unsigned integer.*

Proof. The original Drapper's adder as introduced in [34] is illustrated in fig. 10.

The only quantum gates using the quantum register $|b\rangle$ are the controlled-phase gates. Moreover, they only use the qubits of the right-hand-side register $|b\rangle$ as controls.
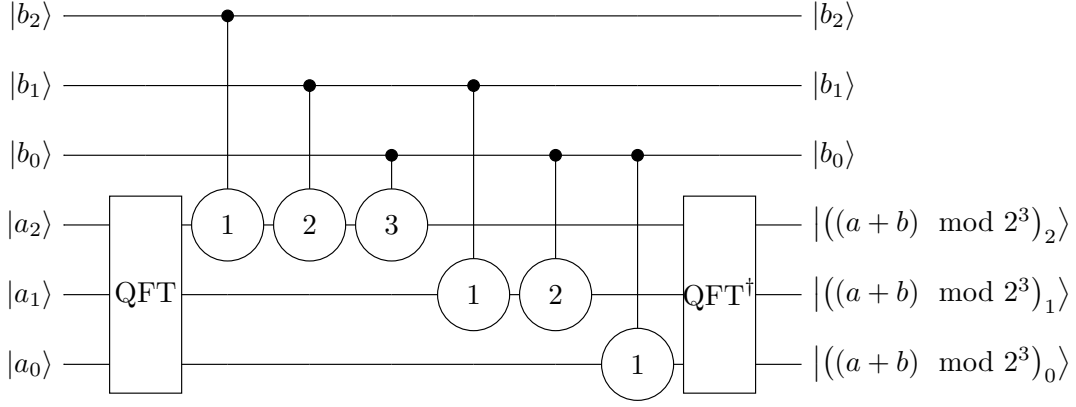
FIG. 10. Original Draper's adder example for 3-qubit registers $|a\rangle$ and $|b\rangle$. The round gates between the two applications of the Quantum Fourier Transform (QFT gates) are controlled phase gates and are defined in [34]. Note that the adder wraps on overflow, meaning that if an overflow happens, the result will be $(a + b) \mod 2^3$.
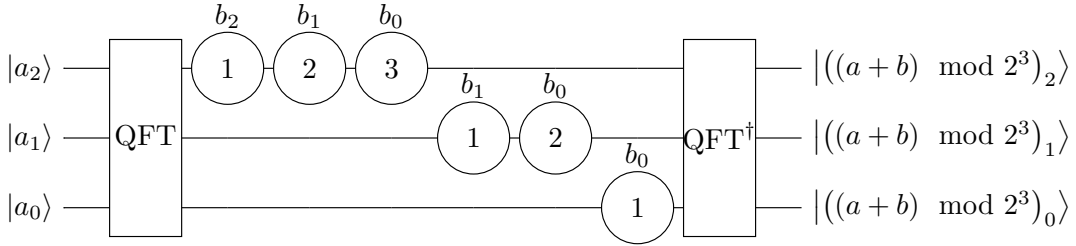

FIG. 11. Modified Draper's adder example for 3-qubit register $|a\rangle$ and 3-bit classical constant b. The round gates between the two applications of the Quantum Fourier Transform (QFT gates) are phase gates and are defined in [34]. A label b_i above a phase gate means that the phase gate should only be applied when the ith bit of b is set to 1. Note that the adder wraps on overflow, meaning that if an overflow happens, the result will be $(a + b) \mod 2^3$.

In the case of a constant value of b known at generation time, we can replace each controlled-phase gate by either a phase gate if the corresponding bit of b is 1 or by an identity gate (or a "no-op" gate) if the bit of b is 0. Once this transformation has been performed, the quantum register $|b\rangle$ is no longer used and can be safely removed from the circuit. $\qquad\square$

The final quantum add gate implementation is depicted in fig. 11, requires $\mathcal{O}\left(n^2\right)$ gates and has a depth of $\mathcal{O}\left(n\right)$. Following [15, 30, 34], the asymptotic gate count can be improved to $\mathcal{O}\left(n \log(n)\right)$ by removing the rotation with an angle below a given threshold that depend on hardware noise.

<center>

c.   The cmp gate

</center>

For the same reasons exposed in the adder implementation in appendix C 3 b, the cmp gate cannot be implemented using the arithmetic comparator presented in [33] because removing the right-hand side qubits seems to be a challenging task.

Instead, we use the idea from [33, Section 4.3] that explain how to implement a comparator only by using a quantum adder. The comparison algorithm works by computing the high-bit of the expression $a - b$. If this high-bit is in the state $|1\rangle$ then $a < b$.

In order to compute the high-bit of $a - b$, several options are open. The two most promising options are described in the following paragraphs.

The first option is to use a subtractor acting on $n + 1$ qubits and behaving nicely on underflows (i.e. underflows result in cycling to the highest-value), as illustrated in fig. 12. This approach requires 2 calls to the subtractor and 1 additional 2-qubit quantum gate.

Another solution would be to use eq. (C8) to change the subtraction into an addition and then use a specialised procedure to compute the high-bit of the addition of two numbers a and b (a being encoded on a quantum register and
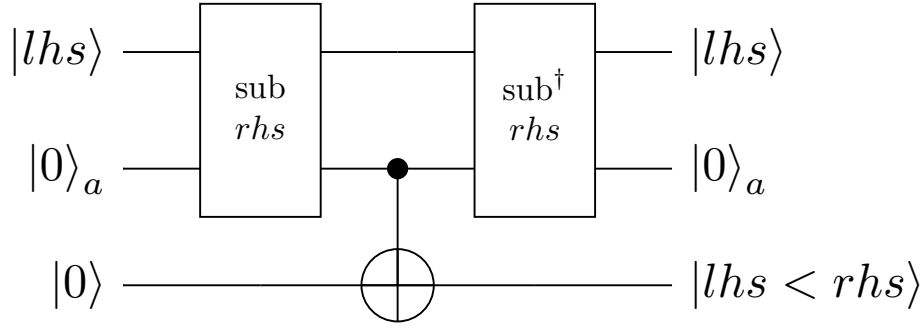
FIG. 12. Computation of the high-bit of $lhs - rhs$ with a $(n+1)$-qubit subtractor. The second quantum register is an ancilla qubit that is appended to the quantum register storing |lhs⟩ in order to form a $(n+1)$-qubit register. The result is stored in a third quantum register as $|1\rangle$ if $lhs < rhs$, else $|0\rangle$.

b a constant). Computing the high-bit of an addition between a quantum register and a constant can be performed with the CARRY gate introduced in [39]. This approach requires $\mathcal{O}(n)$ Toffoli, CNOT and X gates.

Each of the described methods has its advantages and drawbacks.

For example, the first method crucially relies on a quantum subtractor, and will have the same properties as the subtractor used. In our specific case, we use the subtractor implemented with Drapper's adder [34] as explained in appendix C 3 b, which in turn uses the quantum Fourier transform. The main disadvantage of using the QFT when looking at practical implementation on quantum hardware is that the QFT involves phase gates with exponentially small angles. These gates may be implemented correctly up to a given threshold, but very small rotation angles will inevitably not be as precise as *normal* rotation angles due to the hardware limitations in precision. This problem can be circumvented by using an approximate QFT algorithm [15, 30] that will cut all the rotation gates that have a rotation angle smaller than a given threshold from the generated circuit but the algorithm will not be exact anymore (small probability of incorrect result).

On the other hand, the CARRY gate involves only X, controlled-X and Toffoli gates. This restriction makes this implementation more robust than the first one to hardware approximations. Another difference is the connectivity needed by the approaches: the first method relies on a adder implemented with the quantum Fourier transform, which use an all-to-all connectity whereas the CARRY gate, once the qubits correctly ordered, only contains gates on adjacent qubits. As a side note, the exclusive use of logical gates X, controlled-X and Toffoli may allow us to simulate efficiently the CARRY gate on classical hardware as it only involves classical arithmetic.

As a last word, in the future, the QFT may be implemented directly into the hardware chips to make it more efficient because it is one of the most used quantum procedure (and so one of the best candidate for optimisation). Taking this possibility into account seems a little premature right now but may have a high impact on the efficiency and precision of the first solution presented.

After summarising all the drawbacks and advantages, we decided to use the arithmetic comparator for its linear number of gates, because it is based on arithmetic which does not involve exponentially small rotation angles and because the need to have $n-1$ dirty qubits to lend to the procedure is not an issue in our implementation.

d. The eq gate

The last gate the oracle implementation will need is an eq gate, testing the equality between an integer stored in a quantum register and a generation-time constant integer.

This gate has been implemented with a multi-controlled Tofolli gate and a few X gates before and after the control qubits of the Toffoli gates that should be equal to $|0\rangle$. The X gates are necessary because a raw Toffoli gate set its target qubit only when all its control are in the state $|1\rangle$, but we want each control qubit to be equal to a specific bit of the generation-time constant integer, which can be either $|0\rangle$ or $|1\rangle$.

An implementation example is available in fig. 13.

Implementing a NOT gate controlled by n qubits can be done with only one ancilla qubit or $n-2$ garbage qubits and requires $\mathcal{O}(n)$ X, controlled-X or Toffoli gates [1].

FIG. 13. Example of `eq` gate implementation for the compile-time value 5. `X` gates are applied to the second control qubit because the only bit set to 0 in the big-endian binary representation of $5 = 101_2$ is at the second (middle) position.

4.   Oracles for H_1

As noted in appendix C 2, the oracles M_1 and S_1 can be optimized by using the fact that they can encode anything for $|x\rangle_x$ when the xth row of H_1 is empty.

We decided to use this optimization opportunity to add regularity to the description of the H_1 matrix. The *implemented* matrix H_1, denoted as H_1^{impl}, is described in eq. (C9).

Note 6. All indices start at 0. The first row of a matrix has the index 0, the second row the index 1 and so on. This convention is used to match Python's indexing that starts at 0.

$$H_1^{\text{impl}} = \begin{array}{c} N_c \left\{ \begin{array}{c} \\ \\ \\ \\ \end{array}\right. \\ N_c+1 \left\{ \begin{array}{c} \\ \\ \\ \\ \\ \end{array}\right. \\ 2^q-(2N_c+1) \left\{ \begin{array}{c} \\ \\ \\ \\ \\ \\ \end{array}\right. \end{array} \begin{pmatrix} 0 & \cdots & \cdots & 0 & 0 & 1 & 0 & \cdots & 0 & 0 & \cdots & \cdots & \cdots & \cdots & 0 \\ \vdots & & & \vdots & \vdots & \ddots & \ddots & \ddots & \vdots & \vdots & & & & & \vdots \\ \vdots & & & \vdots & \vdots & & \ddots & \ddots & 0 & \vdots & & & & & \vdots \\ 0 & \cdots & \cdots & 0 & 0 & \cdots & \cdots & 0 & 1 & 0 & & & & & \vdots \\ 0 & \cdots & \cdots & 0 & 0 & \cdots & \cdots & \cdots & 0 & \mathbf{0} & & & & & \vdots \\ 1 & \ddots & & \vdots & \vdots & & & & & \vdots & 0 & & & & \\ 0 & \ddots & \ddots & \vdots & \vdots & & & & & \vdots & \vdots & & & & \vdots \\ \vdots & \ddots & \ddots & 0 & \vdots & & & & & \vdots & \vdots & & & & \vdots \\ 0 & \cdots & 0 & 1 & 0 & \cdots & \cdots & \cdots & 0 & 0 & \cdots & \cdots & \cdots & \cdots & 0 \\ 0 & \cdots & \cdots & 0 & \mathbf{0} & 0 & \cdots & \cdots & 0 & 0 & \cdots & \cdots & \cdots & \cdots & 0 \\ \vdots & & & & \ddots & \mathbf{0} & \ddots & & \vdots & \vdots & & & & & \vdots \\ \vdots & & & & & \ddots & \ddots & \ddots & \vdots & \vdots & & & & & \vdots \\ \vdots & & & & & & \ddots & \mathbf{0} & 0 & 0 & & & & & \vdots \\ \vdots & & & & & & & \ddots & \mathbf{0} & 0 & \ddots & & & & \vdots \\ 0 & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & 0 & \mathbf{0} & 0 & 0 & \cdots & \cdots & 0 \end{pmatrix}$$

$$\overbrace{\qquad}^{N_c} \overbrace{\qquad}^{N_c+1} \overbrace{\qquad}^{2^q-(2N_c+1)}$$

(C9)

According to the shape of the matrix in eq. (C9), the oracle M_1 should implement the transformation

$$M_1|x\rangle_x|0\rangle_m \mapsto \begin{cases} |x\rangle_x \otimes |x+(N_c+1)\rangle_m & \text{if } x < (N_c+1) \\ |x\rangle_x \otimes |x-(N_c+1)\rangle_m & \text{else} \end{cases}.$$
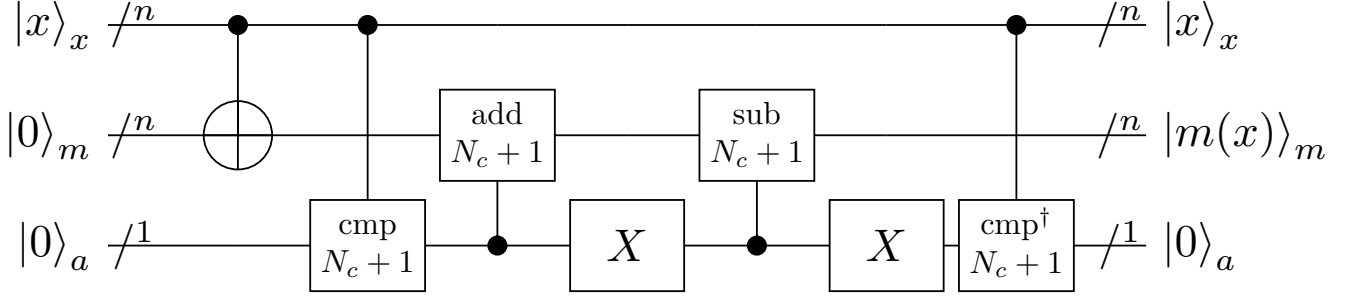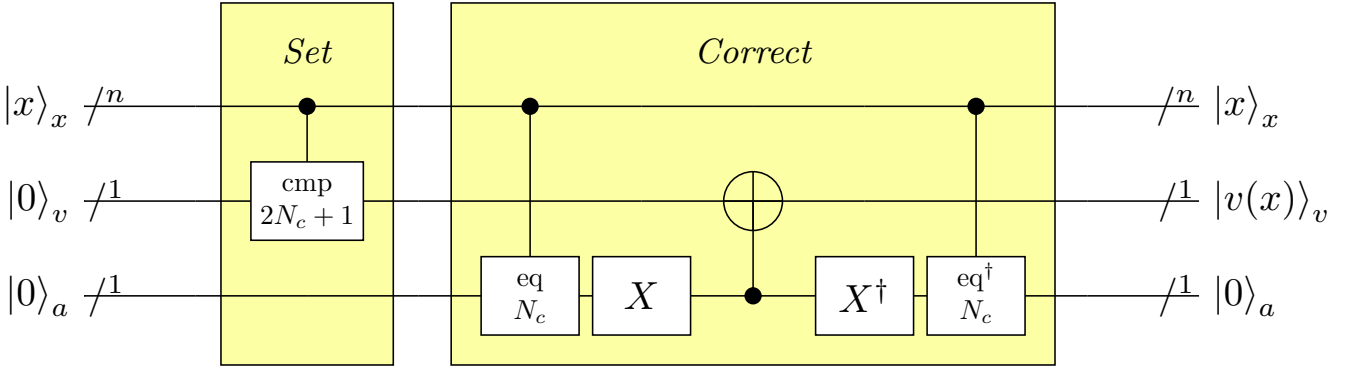
(C10)

FIG. 14. Implementation of the oracle M_1. The `cmp` gate compare the value of the control quantum register (interpreted as an unsigned integer) with the parameter given (written below the `cmp`). If the control register is strictly lower than the parameter, the gate set the qubit it is applied on to $|1\rangle$. The `add` (resp. `sub`) gate used in this quantum circuit add (resp. subtract) the value of its parameter to (resp. from) the quantum register it is applied on only if the control qubit is in the state $|1\rangle$.

FIG. 15. Implementation of the oracle V_1. The `cmp` gate compare the value of the control quantum register (interpreted as an unsigned integer) with the parameter given (written below the `cmp`). If the control register is strictly lower than the parameter, the gate set the qubit it is applied on to $|1\rangle$. The `eq` gate used in this quantum circuit sets its target qubit to $|1\rangle$ if the value of its parameter is equal to the value encoded on the quantum register controling the gate.

M_1 can be easily implemented with the quantum circuit depicted in fig. 14.

The oracle V cannot be simplified using the results from Claim claim 1. It should implement the transformation written in eq. (C11).

$$V_1|x\rangle_x|0\rangle_v \mapsto \begin{cases} |x\rangle_x|1\rangle_v & \text{if } (x < 2N_c + 1) \wedge (x \neq N_c) \\ |x\rangle_x|0\rangle_v & \text{else} \end{cases}. \tag{C11}$$

The implementation of the oracle V_1 is depicted in fig. 15. The first part, *Set*, sets the weight qubit to 1 for all $|x\rangle_x$ such that $x < 2N_c + 1$. As this does not correspond to the correct expression of V, the second part *Correct* is here to set the weight register back to $|0\rangle_v$ when $x == N_c$.

The last oracle left to implement in order to be able to simulate H_1 is S_1, the oracle encoding the signs of the non-zero entries of H_1. The convention used to encode the sign of an entry has been taken from [10] and is: a positive sign is encoded as $|0\rangle_s$, a negative sign is encoded as $|1\rangle_s$. As shown in eq. (C9), H_1 only contains positive non-zero entries so the sign oracle S_1 should implement the simple transformation of eq. (C12): the identity.

$$S_1|x\rangle_x|0\rangle_s \mapsto |x\rangle_x|0\rangle_s \tag{C12}$$

5. Oracles for H_{-1}

The matrix H_{-1} has less regularity than H_1, which will lead to a more complex implementation. The *implemented* matrix H_{-1}, denoted as H_{-1}^{impl}, is described in eq. (C13).

$$
H_{-1}^{\mathrm{impl}} =
\begin{pmatrix}
0 & \cdots & \cdots & 0 & 1 & 0 & \cdots & \cdots & 0 & 0 & \cdots & \cdots & \cdots & \cdots & 0 \\
\vdots & & & \vdots & 0 & -1 & \ddots & & \vdots & \vdots & & & & & \vdots \\
\vdots & & & \vdots & \vdots & \vdots & \ddots & \ddots & \ddots & \vdots & \vdots & & & & \vdots \\
0 & \cdots & \cdots & 0 & 0 & \cdots & 0 & -1 & 0 & \vdots & & & & & \vdots \\
1 & 0 & \cdots & 0 & 0 & \cdots & \cdots & \cdots & 0 & \vdots & & & & & \vdots \\
0 & -1 & \ddots & \vdots & \vdots & & & & & \vdots & & & & & \vdots \\
\vdots & \ddots & \ddots & 0 & \vdots & & & & & \vdots & & & & & \vdots \\
\vdots & & \ddots & -1 & 0 & & & & & \vdots & & & & & \vdots \\
0 & \cdots & \cdots & 0 & -\mathbf{0} & \ddots & & & \vdots & 0 & \cdots & \cdots & \cdots & \cdots & 0 \\
0 & \cdots & \cdots & \cdots & 0 & -\mathbf{0} & \ddots & & \vdots & 0 & \cdots & \cdots & \cdots & \cdots & 0 \\
\vdots & & & & & \ddots & \ddots & \ddots & \vdots & \vdots & & & & & \vdots \\
\vdots & & & & & & \ddots & \ddots & 0 & \vdots & & & & & \vdots \\
\vdots & & & & & & & \ddots & -\mathbf{0} & 0 & & & & & \vdots \\
\vdots & & & & & & & & \ddots & -\mathbf{0} & \ddots & & & & \vdots \\
0 & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & 0 & -\mathbf{0} & 0 & \cdots & \cdots & 0
\end{pmatrix}
\tag{C13}
$$

(with row groupings N_c, N_c+1, $2^q-(2N_c+1)$ and column groupings N_c, N_c+1, $2^q-(2N_c+1)$)

Following the placement of the non-zero and the $\mathbf{0}$ or $-\mathbf{0}$ entries in the matrix H_{-1}^{impl} of eq. (C13), the oracle M_{-1} should implement the transformation

$$
M_{-1}|x\rangle_x|0\rangle_m \mapsto \begin{cases} |x\rangle_x|x+N_c\rangle_m & \text{if } x < N_c \\ |x\rangle_x|x-N_c\rangle_m & \text{else} \end{cases}.
\tag{C14}
$$

This transformation is quite similar to the one implemented by the oracle M_1 in eq. (C10): $N_c + 1$ from the transformation M_1 has been replaced by N_c in the transformation M_{-1}. Thanks to this similarity, the implementation of M_{-1} will be a nearly-exact copy of the implementation of M_1. The full implementation of the M_{-1} oracle is depicted in fig. 16.

The weight oracle V_{-1} is the simplest to implement for the matrix H_{-1}, even if it cannot take advantage of the optimisation discussed in Claim claim 1. The transformation that should be implemented by the oracle V_{-1} is shown in eq. (C15).

$$
V_{-1}|x\rangle_x|0\rangle_v \mapsto \begin{cases} |x\rangle_x|1\rangle_v & \text{if } x < 2N_c \\ |x\rangle_x|0\rangle_v & \text{else} \end{cases}.
\tag{C15}
$$

The implementation of the weight oracle V_{-1} is illustrated in fig. 17.

The last oracle left to implement is S_{-1}, the sign oracle. Due to the sign irregularity in the matrix H_{-1}^{impl}, the implementation of S_{-1} is more involved and requires several ancillary qubits. According to the shape of the matrix H_{-1}^{impl}, the sign oracle S_{-1} should implement the transformation defined in eq. (C16).

$$
S_{-1}|x\rangle_x|0\rangle_s \mapsto \begin{cases} |x\rangle_x|0\rangle_s & \text{if } (x = 0) \vee (x = N_c) \\ |x\rangle_x|1\rangle_s & \text{else} \end{cases}.
\tag{C16}
$$

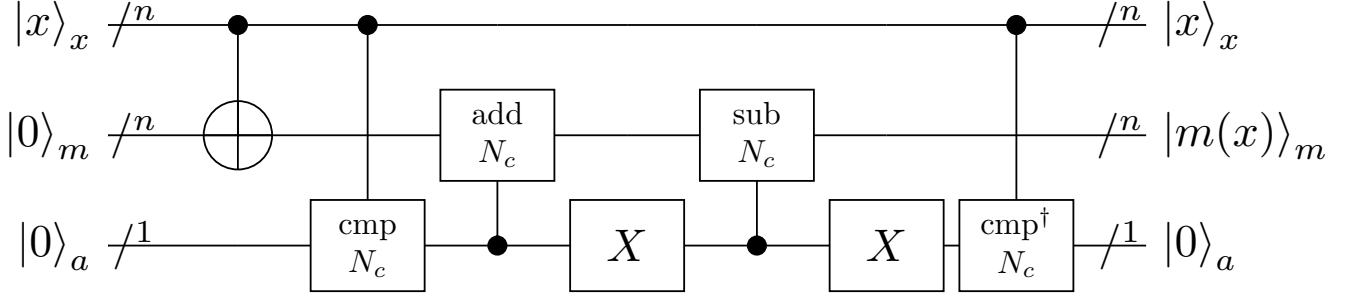An implementation of the oracle S_{-1} is illustrated in fig. 18.

FIG. 16. Implementation of the oracle M_{-1}. The cmp gate compare the value of the control quantum register (interpreted as an unsigned integer) with the parameter given (written below the cmp). If the control register is strictly lower than the parameter, the gate set the qubit it is applied on to $|1\rangle$. The add (resp. sub) gate used in this quantum circuit add (resp. subtract) the value of its parameter to (resp. from) the quantum register it is applied on only if the control qubit is in the state $|1\rangle$.
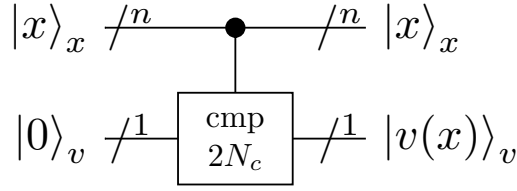

FIG. 17. Implementation of the oracle V_{-1}. The cmp gate compare the value of the control quantum register (interpreted as an unsigned integer) with the parameter given (written below the cmp). If the control register is strictly lower than the parameter, the gate set the qubit it is applied on to $|1\rangle$.

Appendix D: Implementation of higher-order Laplacians

The results shown in this paper have all been generated using the second-order discretisation formula shown in eq. (B2). Higher-order formulas are studied in [32, VI, VII and C].

Note 7. As shown in [32, VII.B], higher-order discretisations with Neumann boundary conditions are not implementable using the algorithm described in appendix B.

In this appendix we replace the second-order formula given in eq. (B2) and used all along the paper by the fourth-
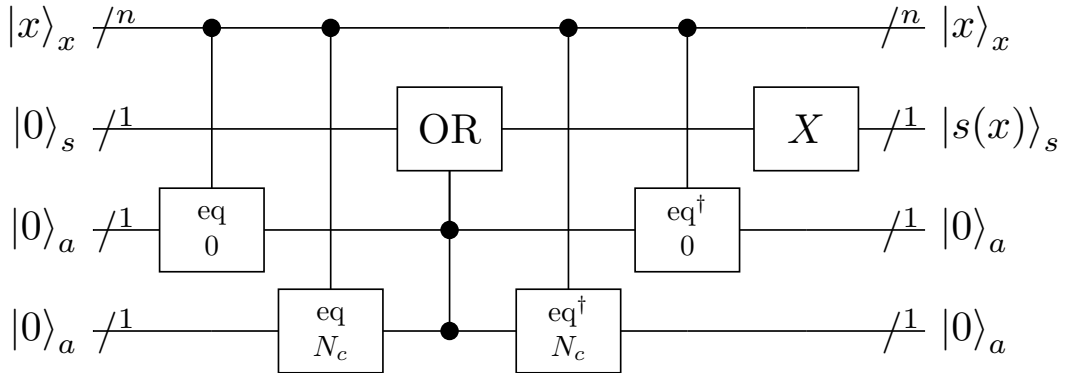

FIG. 18. Implementation of the oracle S_{-1}. The eq gate used in this quantum circuit is presented in fig. 13 and test if the value encoded in its control qubits is equal to the compile-time value given. The OR gate flips the target qubits if and only if at least one of the two control qubits is in the state $|1\rangle$.

order formula given in [32, Eq. (46)] and re-written below

$$\frac{\partial^2 \phi}{\partial x^2}(i\delta x, t) \approx \frac{1}{\delta x^2}\left(\frac{5}{2}\phi_{i,t} - \frac{4}{3}\left(\phi_{i-1,t} + \phi_{i+1,t}\right) + \frac{1}{12}\left(\phi_{i-2,t} + \phi_{i+2,t}\right)\right). \tag{D1}$$

We are left to devise the matrix B_d^4 that satisfy $B_d^4 B_d^{4\dagger} = \Delta_4$ where Δ_4 is the discretisation matrix arising from the fourth-order finite-differences approximation in eq. (D1).

[32, Eq. (47) and VII.C] devised an analytic formula for $\hat{B}_d^4$, the B_d^4 matrix with periodic boundary conditions, using the matrix $\hat{S}$ representing the cyclic permutation $\{1, 2, \ldots, N\}$ with entries $\hat{S}_{i,j} = \delta_{i,(j+1 \mod N)}$ as shown in eq. (D2).

$$\hat{S} = \begin{pmatrix} 0 & 0 & \cdots & \cdots & \cdots & 0 & 1 \\ 1 & 0 & & & & & 0 \\ 0 & \ddots & \ddots & & & & \vdots \\ \vdots & \ddots & \ddots & \ddots & & & \vdots \\ \vdots & & \ddots & \ddots & \ddots & & \vdots \\ \vdots & & & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & \cdots & \cdots & 0 & 1 & 0 \end{pmatrix} \tag{D2}$$

With this definition of $\hat{S}$, the analytic formula for $\hat{B}_d^4$ is given in eq. (D5), with b and c being solution of [32, Eqs. (53,54,55)]. The exact values for b and c are:

$$b = \pm\frac{1}{2\sqrt{3}\sqrt{7 \pm 4\sqrt{3}}} \tag{D3}$$

$$c = \pm\sqrt{\frac{7 \pm 4\sqrt{3}}{12}} \tag{D4}$$

with the $\pm$ signs that can be chosen freely. Note that, $b = \pm\frac{1}{2\sqrt{3}\sqrt{7\pm4\sqrt{3}}}$ and $c = \frac{1}{12b}$ are irrational because $\sqrt{3}\sqrt{7 \pm 4\sqrt{3}} = \sqrt{3}\sqrt{2 + \sqrt{3}}^2 = \sqrt{3}\left(2 + \sqrt{3}\right) = 2\sqrt{3} + 3$ is irrational.

eq. (D6) shows the matrix shape with its entries.

$$\hat{B}_d^4 = c\hat{S} - (b + c) * \mathbb{I} + b\hat{S}^\dagger \tag{D5}$$

$$\approx \begin{pmatrix} b+c & b & 0 & \cdots & \cdots & \cdots & 0 & c \\ c & b+c & b & \ddots & & & & 0 \\ 0 & c & \ddots & \ddots & \ddots & & & \vdots \\ \vdots & & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & & & \ddots & \ddots & \ddots & b & 0 \\ 0 & & & & & \ddots & c & b+c & b \\ b & 0 & \cdots & \cdots & \cdots & 0 & c & b+c \end{pmatrix} \tag{D6}$$

Because periodicity has not been studied in the main use-case of this paper, we would like to also remove the need of periodic boundary conditions in this higher-order laplacian discretisation. This can be achieved by removing the

upper-right entry of $\hat{S}$ by changing it from 1 to 0. The resulting matrix S is shown in eq. (D7).

$$S = \begin{pmatrix} 0 & 0 & \cdots & \cdots & \cdots & \cdots & 0 \\ 1 & 0 & & & & & \vdots \\ 0 & \ddots & \ddots & & & & \vdots \\ \vdots & \ddots & \ddots & \ddots & & & \vdots \\ \vdots & & \ddots & \ddots & \ddots & & \vdots \\ \vdots & & & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & \cdots & \cdots & 0 & 1 & 0 \end{pmatrix} \tag{D7}$$

Using the exact same formula we can devise B_d^4:

$$B_d^4 = cS - (b+c) * \mathbb{I} + bS^\dagger \tag{D8}$$

$$\approx \begin{pmatrix} b+c & b & 0 & \cdots & \cdots & \cdots & \cdots & 0 \\ c & b+c & b & \ddots & & & & \vdots \\ 0 & c & \ddots & \ddots & \ddots & & & \vdots \\ \vdots & & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & & & \ddots & \ddots & b & 0 \\ \vdots & & & & & \ddots & c & b+c & b \\ 0 & \cdots & \cdots & \cdots & \cdots & 0 & c & b+c \end{pmatrix} \tag{D9}$$

Replacing B_d^4 in eq. (B6) we obtain

$$\widetilde{H_d^4} = \frac{1}{\delta x} \begin{pmatrix} 0 & \cdots & \cdots & \cdots & 0 & b+c & b & 0 & \cdots & 0 \\ \vdots & & & & \vdots & c & b+c & b & \ddots & \vdots \\ \vdots & & & & \vdots & 0 & c & \ddots & \ddots & 0 \\ \vdots & & & & \vdots & \vdots & \ddots & \ddots & b+c & b \\ 0 & \cdots & \cdots & \cdots & 0 & 0 & \cdots & 0 & c & b+c \\ b+c & c & 0 & \cdots & 0 & 0 & \cdots & \cdots & \cdots & 0 \\ b & b+c & \ddots & \ddots & \vdots & \vdots & & & & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 & \vdots & & & & \vdots \\ \vdots & \ddots & \ddots & b+c & c & \vdots & & & & \vdots \\ 0 & \cdots & 0 & b & b+c & 0 & \cdots & \cdots & \cdots & 0 \end{pmatrix}. \tag{D10}$$

One of the main difference with the second-order approximation used all along this paper is that, with the fourth-order approximation, the entries of the matrix $\widetilde{H_d^4}$ are no longer multiples of a common number $\alpha \in \mathbb{R}$. This means that we cannot write down $\widetilde{H_d^4}$ as an integer weighted matrix multiplied by a real number, and so the trick used in appendix B 3 to simulate the integer weighted matrix H_d for a time αt is no longer applicable.

Consequently, and independently of the decomposition we use for $\hat{H}_d^4$, at least one of the matrices in the decomposition of $\hat{H}_d^4$ will not be "easy to simulate" as defined in Definition definition 3.

Ultimately, the main consequence of this observation is that we will have to use a real-weighted hamiltonian simulation procedure. Such a procedure can be found in [10] but requires to approximate the real-weighted entries with a fixed-point representation that has at least 2 evident caveats:

1. It is impossible to encode the irrational numbers b and c exactly with a fixed-point representation. This means that we add another layer of approximation, even before the approximation caused by the use of a product-formula.

2. The hamiltonian simulation procedures used for real numbers requires more qubits. More precisely, the number of additional qubits required depends on the desired precision ϵ and grows as $\log_2\left(\frac{1}{\epsilon}\right)$.

Note 8. Even if the H_d^4 matrix seems quite hard to simulate, it is still a 3-sparse matrix. This means that it is still managable to hand-write the oracles. Moreover, having a small number of matrices in the decomposition helps in reducing the error introduced by product-formulas.

<div align="center">

Appendix E: Optimisation of the implementation

</div>

Once the correctness of the implementation validated, one of the most important remaining work is to try to optimise the implementation. The optimisation of a software is often performed as an iterative task.

The first step is to define a figure of merit, a quantity we want to minimise during the optimisation process. Among the most obvious figures of merit are the total number of gates, the number of CNOT gates or the total execution time of the quantum program. More complex quantities can also be considered, such as the execution time using error correction codes or the final state fidelity. In this paper we decided to take into account an estimation of the total execution time of the quantum program on an imaginary device that shares today's chips characteristics.

The second step of the optimisation process consists in isolating the subroutines that contribute the most to the figure of merit. As an example, if the quantum program spend 90% of the total execution time in one subroutine, this subroutine should be the first place to look for optimisations.

After the isolation of one or two subroutines, the actual optimisation can take place. The goal of this third step is to decrease the impact of the subroutines considered on the overall figure of merit without changing the final result of the implementation.

Finally, once the optimisation is performed, the optimisation process can be repeated by re-starting at the second step, until the program is considered sufficiently optimised.

One of the main difficulty we encountered when applying this optimisation process was to correctly isolate the most time-consuming subroutines. In classical computing, this step is usually performed with specialised tools such as `gprof` or a more advanced profiler, but no such tool exist for quantum programs. In order to fill this gap we developed `qprof`, a tool that analyses a quantum program and generates a report similar to the one generated by `gprof`. Using `qprof` and some of the various tools compatible with `gprof`, we plotted the call-graph shown in fig. 19(a).

From this call-graph, it is clear that the adder is *the* most costly subroutine and that it should be optimised. The adder internally uses the Quantum Fourier Transform (QFT), which takes more than 50% of the total execution time. The issue is that the QFT implementation is already very concise and we do not expect to be able to optimise it enough to cut significantly its overall cost. This leads us to the conclusion that a new algorithm that do not require the QFT should be used to implement an adder. Such an algorithm can be found in [64].

Changing the implementation of the adder from Draper's adder to the arithmetic-based adder from [64] improves drastically the total execution time of the quantum program and produce the call-graph in fig. 19(b).

<div align="center">

Appendix F: Gate count analysis

1. Precise subroutines gate counts

</div>

table I summarise the gate count and ancilla qubit requirements for all the major subroutines used in the wave equation solver implementation. Using the entries of this table, it is possible to compute an estimation of the number of gates required to solve the wave equation.

As explained in appendix A, we need to simulate each of the 1-sparse Hamiltonians in the decomposition. Aggregating the estimates in table I we obtain the costs in table II for the Hamiltonian simulation part. Note that the cost of the adder has been voluntarily omited from the computations in order to be able to compare the cost with different adder implementations. Let a be the gate cost of the adder implementation chosen, the cost of simulating the Hamiltonian needed to solve the 1-dimensional wave equation is: $82n - 35 + 12a_{\texttt{Toffoli}}$ Toffoli gates, $84n + 21 + 9\,[0, n-1] + 12a_{\texttt{CNOT}}$ CNOT gates and $\mathcal{O}(n) + 12a_{1-\texttt{qubit}}$ 1-qubit gates.

Choosing an adder implementation and simplifying the gate counts by omitting negligible terms we obtain the gate counts summarised in table III. It is interesting to note that even if the arithmetic-based adder adds huge constants in the gate count, it does not change the asymptotic complexity whereas Draper's adder changes the number of CNOT gates required from $\mathcal{O}(n)$ to $\mathcal{O}(n^2)$.
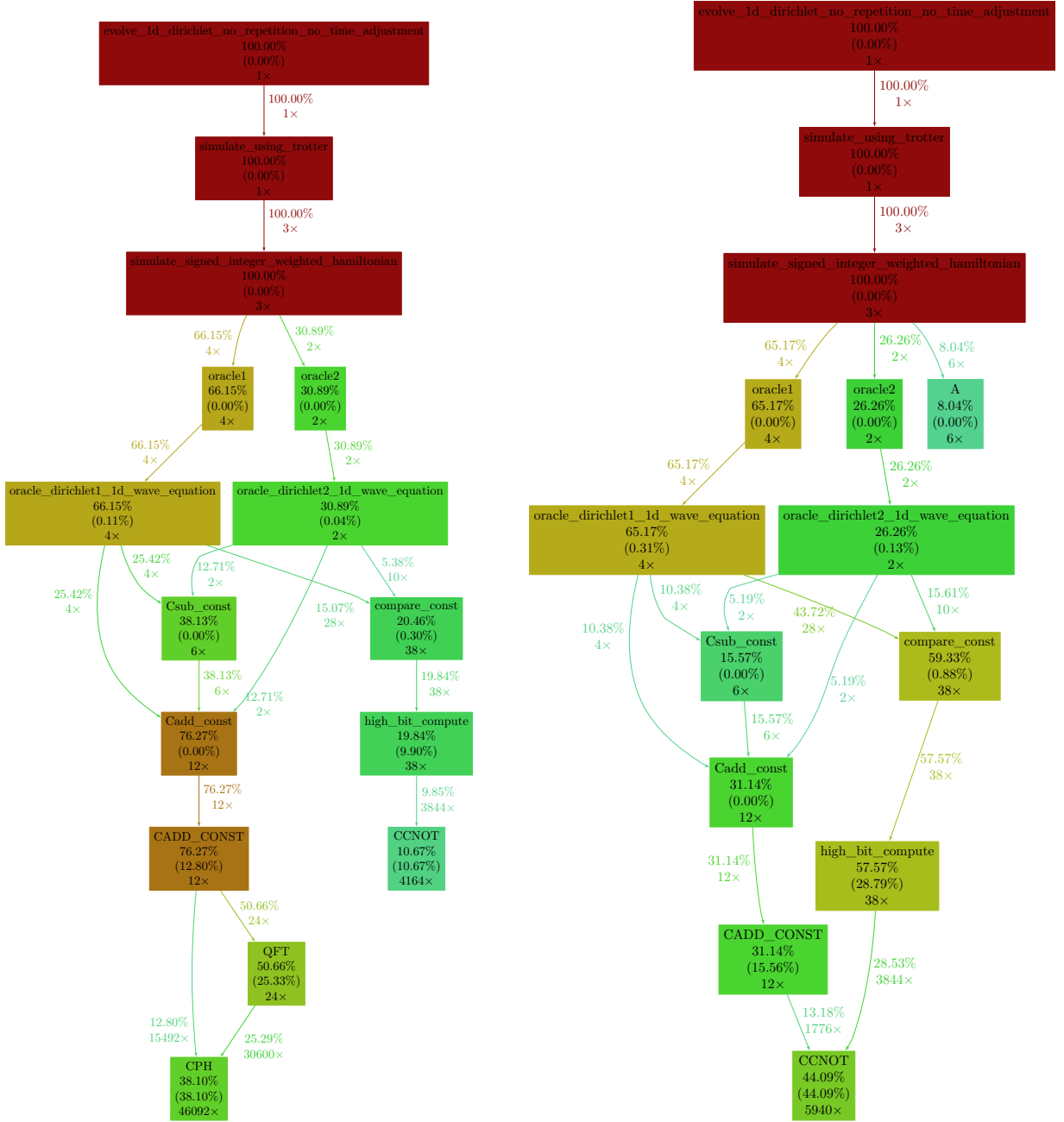
(a)Call graph of the quantum wave equation solver using Draper's adder (QFT-based).

(b)Call graph of the quantum wave equation solver using an arithmetic-based adder.

FIG. 19. All the gates or subroutines that account for less than 5% of the total execution time are not displayed. Execution times for $u1$, $u2$, $u3$ and cx gates have been averaged over all the data available for the quantum chip IBMQ Melbourne. Using the arithmetic-based adder, the overall execution time improved by a factor of 31.

2. Impact of the precision requirements

The gate counts presented in table I, table II and table III are only valid when the precision of the solver is not accounted for. When the solver precision matters, an additional step that consists is splitting the Hamiltonian Simulation into r steps needs to be performed as noted in [27, arXiv: Appendix F].

Several bounds exist to determine a $r \in \mathbb{N}^*$ that will analytically ensure that the maximum allowable error ϵ is not exceeded. The definition of such bounds can be found in [27, arXiv: Appendix F] and [28].

Gate	Toffoli count	CNOT count	1-qubit gate count	# ancillas	notes
`or`	1	0	5	0	
`QFT`	0	$3\left(2n^2 - 2n + \lfloor\frac{n}{2}\rfloor\right)$	$2\left(n^2+n\right)\ H$ $4\left(n^2-n\right)\ T$ $\frac{n^2-n}{2}\ R_n$	1 $|0\rangle$-init	R_n gates might need to be decomposed [53].
`add_arith`	$20n-10$	$22n$	0	$n-1$ $|0\rangle$-init	See [64].
`add_qft`	0	$6\left(2n^2 - 2n + \lfloor\frac{n}{2}\rfloor\right)$	$2\left(n^2+n\right)\ H$ $4\left(n^2-n\right)\ T$ $3\frac{n^2-n}{2}\ R_n$	1 $|0\rangle$-init	See `QFT` note on R_n. fig. 11.
`sub_qft`	0	$6\left(2n^2 - 2n + \lfloor\frac{n}{2}\rfloor\right)$	$2\left(n^2+n\right)\ H$ $4\left(n^2-n\right)\ T$ $3\frac{n^2-n}{2}\ R_n$ $2n\ X$	1 $|0\rangle$-init	See `QFT` note on R_n. fig. 9.
`CARRY`	$2(n-1)$	$2 + [0, n-1]$	$2n + [0, n-1]\ X$	$n-1$ borrowed	See [39].
n-contr. `CNOT`	$4n$	0	0	n borrowed	See [1].
`eq`	$4n$	0	$2\,[0,n]\ X$	n borrowed	fig. 13.
`cmp`	$2\left(n-1\right)$	$2 + [0, n-1]$	$4n + [0, n-1]\ X$	$n-1$ borrowed	See `CARRY` and appendix C 3 c.
`A`	$2n$	$4n$	$3n\ H\ \ 3n\ S$ $2n\ T\ \ 2n\ X$	0	See [10, Fig. 4.3.].
$e^{-iZ\otimes Z\otimes Ft}$	$8n$	$24n$	$36n\ P_h$ $8\ X$	0	Adapted from [10, Fig. 4.6]
`1-sparse HS`	$10n$	$28n$	$3n\ H\ \ \ \ \ 3n\ S$ $2n\ T\ \ \ 2n+8\ X$ $36n\ P_h$	0	Oracle implementation cost not included. 2 calls to the oracle are required. fig. 7.
M_1	$4\left(n-1\right)$	$5 + 2\,[0, n-1]$	$10n + 2 + [0, n-1]\ X$	1 $|0\rangle$-init $n-1$ borrowed	`add` implementation cost not included. 2 calls to `add` are required. fig. 14.
V_1	$2\left(n-1\right)$	$2 + [0, n-1]$	$4n + [0, n-1]\ X$	$n-1$ borrowed	fig. 15.
S_1	0	0	0	0	eq. (C12).
M_{-1}	$4\left(n-1\right)$	$5 + 2\,[0, n-1]$	$10n + 2 + [0, n-1]\ X$	1 $|0\rangle$-init $n-1$ borrowed	`add` implementation cost not included. 2 calls to `add` are required. fig. 16.
V_{-1}	$2\left(n-1\right)$	$2 + [0, n-1]$	$4n + [0, n-1]\ X$	$n-1$ borrowed	fig. 17.
S_{-1}	$16n+1$	0	$5 + 8\,[0,n]\ X$	n borrowed	fig. 18.

TABLE I. Precise gate count for the most important subroutines used in the quantum implementation of the wave equation solver. n always represnent the size of the input(s), except for the n-controlled `CNOT` where n is the number of controls. When the number of gates depends on a generation-time value, the range of all the integer values possible is shown with square brackets. For example, $[0, n-1]$ means that, depending on the generation-time value provided, the number of gates will be an integer between 0 and $n-1$ included. $|0\rangle$-init ancillas represent the standard ancilla-type: qubits that are given in the state $|0\rangle$ and should be returned in that exact same state. On the other side, borrowed ancillas can be given in any state and should be returned in the exact same state they were borrowed in.

Unitary	Toffoli count	CNOT count	1-qubit gate count	# ancillas	notes
$e^{-iH_1 t}$	$22n - 12$	$28n + 7 + 3[0, n-1]$	$3n\ H \quad 3n\ S$ $2n\ T \quad 36n\ P_h$ $30n + 10 + 2[0, n-1]\ X$	$1\ \lvert 0 \rangle$-init $n-1$ borrowed	**add** implementation cost not included. 4 calls to **add** are required.
$e^{-iH_{-1} t}$	$38n - 11$	$28n + 7 + 3[0, n-1]$	$3n\ H \quad 3n\ S$ $2n\ T \quad 36n\ P_h$ $30n + 15 + 10[0, n]\ X$	$1\ \lvert 0 \rangle$-init $n-1$ borrowed	**add** implementation cost not included. 4 calls to **add** are required.
e^{-iHt}	$82n - 35$	$84n + 21 + 9[0, n-1]$	$9n\ H \quad 9n\ S$ $6n\ T \quad 108n\ P_h$ $90n + 35 + 14[0, n]\ X$	$1\ \lvert 0 \rangle$-init $n-1$ borrowed	**add** implementation cost not included. 12 calls to **add** are required.

TABLE II. Number of gates and ancillas needed to simulate the easy-to-simulate Hamiltonians H_1 and H_{-1} that are part of the decomposition of H as well as e^{-iHt}. It is important to realise that the gate counts for e^{-iHt} are only valid up to a given t or ϵ (once one is fixed, the value of the other can be computed). In order to make the gate count generic for any t and ϵ, the number of repetitions should be computed (see n in eq. (A5)). Note that some of the $[0, n-1]$ ranges have been simplified to $[0, n]$ for conciseness.

Adder used	Toffoli count	CNOT count	1-qubit gate count	# ancillas
`add_qft`	$82n - 35$	$144n^2 - 60n$	$24n^2 + 25n\ H \quad 9n\ S$ $48n^2 - 42n\ T \quad 108n\ P_h$ $18n^2 - 18n\ R_n$ $114n + 35 + 14[0, n]\ X$	$2\ \lvert 0 \rangle$-init $n-1$ borrowed
`add_arith`	$222n - 175$	$348n + 21 + 9[0, n-1]$	$9n\ H \quad 9n\ S$ $6n\ T \quad 108n\ P_h$ $90n + 35 + 14[0, n]\ X$	$n\ \lvert 0 \rangle$-init $n-1$ borrowed

TABLE III. Number of gates and ancillas needed to simulate the Hamiltonian used to solve the 1-dimensional wave equation depending on the adder implementation used. It is important to realise that the gate counts for e^{-iHt} reported in this table are only valid up to a given t or ϵ (once one is fixed, the value of the other can be computed). In order to make the gate count generic for any t and ϵ, a number of repetitions r should be computed (named n in eq. (A5) and studied in [27, arXiv: Appendix F] and [28]). Note that the gate counts have been simplified by removing negligible terms when possible.

The first bound has been devised by analytically bounding the error of simulation due to the Trotter-Suzuki formula approximation by ϵ_0

$$\left\lVert \exp\left(-it \sum_{j=0}^{m-1} H_j \right) - \left[S_{2k}\left(-\frac{it}{r} \right) \right]^r \right\rVert \leqslant \epsilon_0 \tag{F1}$$

and then let $\epsilon_0 \leqslant \epsilon$ for a given desired precision ϵ. If we let $\Lambda = \max_j \lVert H_j \rVert$ and

$$\tau = 2m5^{k-1}\Lambda \lvert t \rvert \tag{F2}$$

then

$$r_{2k}^{ana} = \left\lceil \max\left\{ \tau, \sqrt[2k]{\frac{e\tau^{2k+1}}{3\epsilon}} \right\} \right\rceil. \tag{F3}$$

This bound is called the *analytic bound.*

A better bound called the *minimised bound* can be devised by searching for the smallest possible r that satisfies the conditions detailed in [27, Propositions F.3 and F.4]. This bound is rewritten in Equation (F4).

$$r_{2k}^{min} = \min\left\{ r \in \mathbb{N}^* : \frac{\tau^{2k+1}}{3r^{2k}} \exp\left(\frac{\tau}{r}\right) < \epsilon \right\} \tag{F4}$$

Another bound involving nested commutators of the H_i is described in [28] and gives

$$r_{2k}^{comm} \in \mathcal{O}\left(\frac{\alpha_{comm}^{\frac{1}{2k}} t^{1+\frac{1}{2k}}}{\epsilon^{\frac{1}{2k}}} \right) \tag{F5}$$

where k is the order of the product-formula used, t the time of simulation, ϵ the error and

$$\alpha_{comm} = \sum_{i_0, i_1, \ldots, i_p = 0}^{m-1} \left\| \left[H_{i_p}, \ldots \left[H_{i_1}, H_{i_0} \right] \right] \right\|. \tag{F6}$$

Once the value of r has been computed, the quantum circuit simulating the matrix H for a time $\frac{t}{r}$ should be repeated r times. This adds a factor of r in front of all the gate counts computed in table I, table II and table III.

3. Impact of error-correction

When error-correction is studied, two gates are particularly important: T and Toffoli gates. The T gate has a prohibitive cost when compared to the Clifford quantum gates and implementing a Toffoli gate requires 7 of such T gates as noted in [36] and [56, Fig. 1].

table IV summarise the cost of the non Clifford quantum gates used in the implementation of the 1-dimensional wave equation solver. The rotation gates need to be approximated. One solution to approximate the R_n and P_h gates is given in [53]. In order to obtain practical results as opposed to theoretical ones, we chose to use the number computed in [45, Table 1].

The final T-count is summarised in fig. 20. From fig. 20(b) it is clear that the add_arith implementation is more efficient than the add_qft one.

Gate	T count	Notes
T	1	
S	2	
CCNOT	7	See [36].
P_h	379	$\epsilon = 10^{-15}$, approximated from [45].
R_n	379	$\epsilon = 10^{-15}$, approximated from [45].

TABLE IV. T-gate cost of the non Clifford quantum gates used in the wave equation solver implementation.

Adder used	T-count
add_qft	$6870n^2 + 34660n - 245$
add_arith	$42510n - 1225$

(a)Number of T-gates needed to simulate the Hamiltonian used to solve the 1-dimensional wave equation depending on the adder implementation used. Based on table III and table IV.
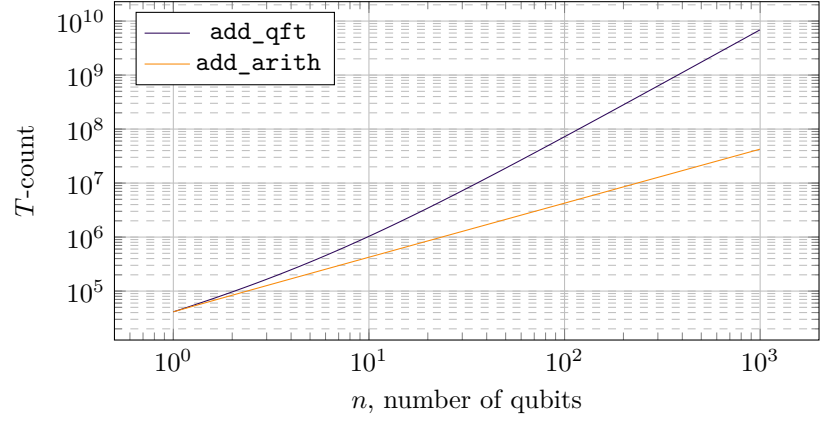
(b)Plot of the T-count devised in fig. 20(a) for the two different adder implementations.

FIG. 20. Analysis of the T-count of the 1-dimensional wave equation solver quantum implementation with respect to the adder implementation used.