Computer Physics Communications 248 (2020) 107001

Contents lists available at ScienceDirect

Computer Physics Communications

journal homepage: www.elsevier.com/locate/cpc

Quantum circuits synthesis using Householder transformations ☆

Timothée Goubault de Brugière [a,c,*], Marc Baboulin [a], Benoît Valiron [b], Cyril Allouche [c]

[a] Université Paris-Sud, Laboratoire de Recherche en Informatique, Orsay, France
[b] École CentraleSupélec, Laboratoire de Recherche en Informatique, Orsay, France
[c] Atos Quantum Lab, Les Clayes-sous-Bois, France

ARTICLE INFO

Article history:
Received 21 December 2018
Received in revised form 5 September 2019
Accepted 22 October 2019
Available online 13 November 2019

Keywords:
Quantum computing
Quantum circuit synthesis
QR factorization
Householder transformations
Multicore and GPU computing

ABSTRACT

The synthesis of a quantum circuit consists in decomposing a unitary matrix into a series of elementary operations. In this paper, we propose a circuit synthesis method based on the QR factorization via Householder transformations. We provide a two-step algorithm: during the first step we exploit the specific structure of a quantum operator to compute its QR factorization, then the factorized matrix is used to produce a quantum circuit. We analyze several costs (circuit size and computational time) and compare them to existing techniques from the literature. For a final quantum circuit twice as large as the one obtained by the best existing method, we accelerate the computation by orders of magnitude.

1. Introduction

In the 1980s the notion of a quantum computer emerged as a response to the announced limitation of conventional computers in terms of computing power. Feynman [1], then Deutsch [2] announced and theorized the first foundations of this new paradigm that must override our current machines. Ten years later, we saw the first concrete algorithms capable of achieving this quantum supremacy: the Grover algorithm theoretically enables us to search into an unstructured database quadratically faster than in the conventional case [3] and the Shor algorithm is expected to be able to break RSA, jeopardizing the security of current encryption tools [4,5]. Quantum computing is now a research topic of growing interest and many algorithms are designed in numerous fields to try to surpass classical computers. Examples are various: machine learning [6,7], linear algebra [8–10], backtracking algorithms [11] or even combinatorial optimization [12]. The interaction between classical computing and quantum computing is also studied, leading to hybrid quantum-classical computers [13]. Behind all these new algorithms lies a common formalism: the quantum circuit. Developed by Yao [14], the concept of a quantum circuit remains so far the preferred way to describe quantum algorithms. Similarly to the compilation in classical computing, transforming a high level concept – or more generally a concept unknown to the hardware – into a sequence of basic instructions for the machine is a central problem. In quantum computing, everything can be modeled with notions of linear algebra: states are vectors, and operators are unitary matrices. The compilation problem can be formalized as the transformation of a unitary matrix into a quantum circuit consisting of elementary (unitary) operations admissible by the hardware and referred to as elementary quantum gates. The development of quantum algorithmics has fostered the emergence of high-level languages [15–17] to efficiently describe and program concrete instances of quantum algorithms. With the limited resources that are going to be available at first for quantum computers, it is crucial to design an automated compilation process minimizing the classical and quantum resources used by a given quantum program.

When turning a unitary matrix into a quantum circuit, several aspects have to be considered. First, one has to decide on the set of admissible elementary operations. Then, one has to choose the resources to be minimized: are we interested in the smallest possible circuit, or are we also considering the classical resources used to produce the circuit and the time required to do so? The former problem is very theoretical and math-oriented. An operator acting on n qubits is represented by a matrix of size $2^n \times 2^n$. Generating a circuit from an arbitrary matrix is therefore a problem that scales exponentially in n in general, and the problem of finding the smallest possible circuit for a particular operator remains challenging [18]. Nonetheless, several techniques have been developed to this end using, e.g., decomposition methods [19–23]. The resulting number of gates however still lies within a factor of 2 of the theoretical lower bound [24]. We are currently in the NISQ (Noisy Intermediate-Scale Quantum) era [25]: the quantum hardware is noisy and it

☆ The review of this paper was arranged by David W. Walker.

* Corresponding author at: Université Paris-Sud, Laboratoire de Recherche en Informatique, Orsay, France.
E-mail addresses: timothee.goubault@lri.fr (T. Goubault de Brugière), marc.baboulin@lri.fr (M. Baboulin), benoit.valiron@lri.fr (B. Valiron), cyril.allouche@atos.net (C. Allouche).

https://doi.org/10.1016/j.cpc.2019.107001
0010-4655/

is hard to perform long computations. In this paper we foresee the future of the NISQ era where full fault tolerant quantum computation will be available. With the advent of such systems, we believe that the synthesis of generic operators on small to medium register size will become critical. For example, one can already get a glimpse of such issue in quantum machine learning problems [6]. Meanwhile, we can also rely on post processing methods that integrate the presence of noise in the hardware and make the connection between ideal quantum circuit synthesis and the hardware constraints [26].

Instead of only focusing on the size of the circuit, one can consider the problem in its globality and also take into account the quantity of classical resources needed, and in particular the time it takes to generate the circuit. Such optimization is particularly useful, e.g. when one has to compile a continuous stream of quantum circuits on the fly or when the quantum operator is parameterized and one has to recompile the parameters of the resulting quantum circuit every time the operator changes. Improving the compilation time also allows to reach larger problem sizes. This aspect of optimization is a recent topic of research [27–30] and is the focus of our paper.

1.1. Contributions

The main contributions of the paper are as follows.

- We adapt the well known and numerically stable QR factorization based on Householder transformations [31] to the factorization of unitary matrices. The adaptation heavily relies on the specific structure of unitary matrices. We exhibit a significant theoretical and practical speedup of our specific QR algorithm compared to the unmodified QR routine and the usual technique for quantum circuit synthesis based on the quantum Shannon decomposition (QSD) [23].
- We propose a complete circuit synthesis method using this specialized QR decomposition with a complexity analysis for circuit size and arithmetical operations. If some existing theoretical and experimental works for quantum circuits synthesis with Householder transformations have been undertaken [32–34], to our knowledge none has proposed an implementation method and a final circuit construction with clearly defined properties. Overall, our technique is faster than the QSD-based method while providing circuits twice as large.[1]
- We backup our approach with benchmarks on multicore and GPU architectures for random unitary matrices operating on up to 15 qubits.

1.2. Plan of the paper

The plan of this paper is as follows. In Section 2 we give some background about quantum computing, quantum circuits and the issues in quantum compilation. Then we detail the new adapted Householder algorithm in Section 3 and we explain in Section 4 how to convert this factorization into a quantum circuit. Section 5 presents the performance obtained on multicore and GPU architectures by our algorithm. We also compare our results with a reference algorithm based on the Quantum Shannon Decomposition method. We conclude in Section 6.

[1] This extra cost in the final quantum circuit is not negligible, especially when considering the current limitations of the quantum hardware. It may be possible that the gain in the classical process will not compensate the execution time of the twice as large quantum circuit on real hardware. However, we can handle problem sizes that were unreachable before with the QSD, regardless of the quality of the hardware. We believe our approach highlights the tradeoffs between two measures of complexity (circuit size/compilation time) and that this has to be taken in consideration when synthesizing generic quantum circuits.

1.3. Notations

Throughout this paper we will use the following notations. $\mathcal{U}(k)$ denotes the set of unitary matrices of size k, i.e. $\mathcal{U}(k) = \{M \in \mathbb{C}^{k\times k} \mid M^\dagger M = I\}$, where I is the identity matrix and $M^\dagger$ is the conjugate transpose of the matrix M. The notation $\|\cdot\|$ refers to the Euclidean norm of a vector and e_i is the ith canonical vector. The term *flops* stands for *floating-point operations* and the flop count evaluates the volume of work in a computation. Unless otherwise specified these flops are given in complex arithmetic. The linear algebra formulas will be presented using matlab-like notations.

2. Background

The core of quantum computation consists in encoding information on the state $|\phi\rangle$ of a quantum system. The computational model is derived from the laws of quantum mechanics: the state $|\phi\rangle$ is represented by a normalized column vector in a (finite dimensional) Hilbert space $\mathbb{C}^k$. The allowed transformations one can perform on $|\phi\rangle$ can be derived from the Schrödinger equation. In this paper we focus on *unitary* transformations. A quantum computation acting on the vector $|\phi\rangle \in \mathbb{C}^k$ is therefore in this paper regarded as a unitary matrix $U \in \mathcal{U}(k)$. After computation, the resulting state is $U|\phi\rangle$. A *sequential* application of two transformations U and V yields the state $V(U|\phi\rangle) = (VU)|\phi\rangle$ and corresponds to a matrix multiplication.

The basic unit of information in quantum computation is the quantum bit, also called qubit. It is encoded by a two-level quantum system (e.g., the spin of an electron) whose state can be in a linear superposition of both levels – called the basis states – according to the laws of quantum mechanics. We usually write $|0\rangle$ to represent the first basis state and $|1\rangle$ the second one (to follow the analogy with the classical case). The general form $|\psi\rangle$ of the state of a qubit is then the linear combination of these basis elements $|0\rangle$ and $|1\rangle$ (also called "superposition"):

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$$

where α, β are complex numbers such that $|\alpha|^2 + |\beta|^2 = 1$. In other words, the state of a qubit is mathematically equivalent to a unit vector $\binom{\alpha}{\beta} \in \mathbb{C}^2$ and the basis states are the usual basis vectors

$$|0\rangle = \begin{pmatrix}1\\0\end{pmatrix} \quad |1\rangle = \begin{pmatrix}0\\1\end{pmatrix}.$$

The qubit is not the only logical unit possible in quantum computing: using 3-level systems (by adding the basic state $|2\rangle$) one can manipulate qutrits; more generally with a d-level system we talk about qudits. However the research in quantum computing today uses mostly qubits.

The state of the quantum system consisting of the *combination* of two systems A and B resides in the Kronecker (tensor) product of the space of states of A and the space of states of B. In particular, to encode n qubits, one can use n two-level systems that together can be seen as a single 2^n level system. The evolution of this system is governed by the left multiplication by unitary matrices in $\mathcal{U}(2^n)$. The basis vectors of the space $\mathbb{C}^{2^n}$ are of the form $|x_1\rangle \otimes \cdots \otimes |x_n\rangle$ with $x_i = 0$ or 1. The usual ordering of the basis states corresponds to the lexicographic order. For example, in the case of two qubits the basis states are

$$|0\rangle \otimes |0\rangle = \begin{pmatrix}1\\0\\0\\0\end{pmatrix}, \quad |0\rangle \otimes |1\rangle = \begin{pmatrix}0\\1\\0\\0\end{pmatrix},$$
$$|1\rangle \otimes |0\rangle = \begin{pmatrix}0\\0\\1\\0\end{pmatrix}, \quad |1\rangle \otimes |1\rangle = \begin{pmatrix}0\\0\\0\\1\end{pmatrix}.$$

Table 1
Usual elementary unitary matrices for representing quantum gates.

$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$	$\begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$
X	Y	Z	CNOT	SWAP
$\frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix}$	$\begin{pmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{pmatrix}$		
H	S	T		

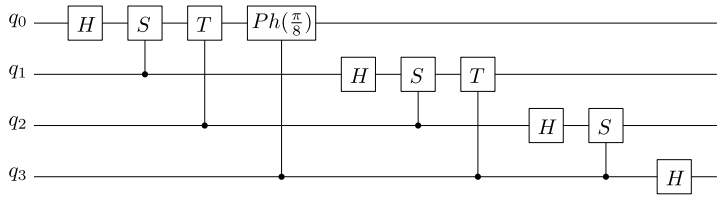

Fig. 1. Quantum circuit for the Quantum Fourier Transform.

To combine operators acting on distinct subsystems, we again use the tensor product. If $|\psi\rangle$ (resp. $|\phi\rangle$) is an n-qubit (resp. m-qubit) state and one applies an operator A on $|\psi\rangle$ (resp. B on $|\phi\rangle$) then using the global system on $n+m$ qubits it is equivalent to applying the operator $A \otimes B$ on the state $|\psi\rangle \otimes |\phi\rangle$, where $\otimes$ denotes the Kronecker product [35].

When a state on n qubits cannot be written as a tensor product of two substates then the state is said to be entangled. The Bell states are simple examples of entangled states on two qubits, one of them is defined by

$$|\Phi\rangle = \frac{1}{\sqrt{2}}\left(|0\rangle \otimes |0\rangle + |1\rangle \otimes |1\rangle\right)$$

and one can check that it cannot be expressed as the tensor product of two one-qubit states. Entanglement is believed to be a key in the quantum supremacy over classical computation [36] and research is performed to better understand its role, for example by giving a measure of how entangled a state is [37]. An operator that can produce entangled states is said to be an *entangling* operator.

Beside composition and combination, a third operation is usually considered: an operation can be *controlled*. If $M \in \mathcal{U}(2^n)$ is an operator acting on n qubits, there are two canonical operations on $n+1$ qubits: the positively-controlled-M defined as the block matrix $\left(\begin{smallmatrix} I & 0 \\ 0 & M \end{smallmatrix}\right)$ and the negatively-controlled-M, defined as $\left(\begin{smallmatrix} M & 0 \\ 0 & I \end{smallmatrix}\right)$. Both block-matrices are operators in $\mathcal{U}(2^{n+1})$. The former sends $|0\rangle \otimes |\phi\rangle$ to $|0\rangle \otimes |\phi\rangle$ and $|1\rangle \otimes |\phi\rangle$ to $|1\rangle \otimes (M|\phi\rangle)$. The latter does the opposite: it sends $|0\rangle \otimes |\phi\rangle$ to $|0\rangle \otimes (M|\phi\rangle)$ and $|1\rangle \otimes |\phi\rangle$ to $|1\rangle \otimes |\phi\rangle$.

An important notion is the *preparation* and *de-preparation* of states. Preparing a state $|\Phi\rangle$ consists in applying an operator U to the state $|0\rangle$ to obtain the state $|\Phi\rangle$. Conversely, the de-preparation of the state $|\Phi\rangle$ consists in applying $U^\dagger = U^{-1}$ to obtain the state $|0\rangle$.

Quantum gates. Though the theory allows arbitrary unitary matrices, the physical hardware is usually only capable of handling a fixed set of unitary matrices operating on one or two qubits. These elementary matrices are called *quantum gates*, and we can mention the following (see Table 1):

- the Pauli operators X, Y, Z (the X gate is equivalent to the classical NOT gate),
- the Hadamard gate H which enables to transform a pure state (i.e. $|0\rangle$ or $|1\rangle$) into an equal superposition of $|0\rangle$ and $|1\rangle$,
- the continuous set of elementary rotations R_x, R_y, R_z defined by

  $R_G(\alpha) = \cos(\alpha/2)I_2 - i\ \sin(\alpha/2)G$ with $G \in \{X, Y, Z\}$

  where X, Y, Z are the Pauli operators and i is the unit imaginary number.
- the continuous set of phase gates defined by

  $$Ph(\theta) = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\theta} \end{pmatrix}$$

  adding a phase to the state $|1\rangle$; among this set two gates are of particular use: the gate T ($\theta = \pi/4$) and the gate S ($\theta = \pi/2$). Note that $Ph(\theta)$ is simply $R_z(\theta)$ modulo a global phase $e^{-i\frac{\theta}{2}}$.

Amongst the frequently used 2-qubit gates, one can name the CNOT-gate, which is the positively-controlled X-gate, and the SWAP gate, flipping the state of two qubits. Other examples of commonly encountered gates are controlled-rotations with arbitrary angles.

Quantum circuit. The usual graphical language for representing composition and combination of operator is the equivalent of the boolean circuit for classical computing: the quantum circuit. A quantum circuit consists in a series of parallel, horizontal wires on which are attached boxes. Each wire corresponds to a qubit, and vertical combination corresponds to the Kronecker (tensor) product. The circuit is read from left to right and each box corresponds to a quantum gate (i.e. a unitary operator) applied on the corresponding qubits. Controlled-gates have a special representation: the controlling qubit is represented with a bullet if the control is positive and a circle if the control is negative. A vertical line then connects the controlling qubit to the gate to be controlled. The notation easily extends to multiple controls.

As an example of quantum circuit combining several gates together, the so-called Quantum Fourier Transform [38] is represented in Fig. 1. It enables us to visualize the use of elementary gates: H, S, T, phase-gate, and positive controls.

Universality. We say that a set of gates is universal if any quantum operator, acting on any number of qubits, can be implemented as a sequence of gates from this set (see e.g. [38, Sec. 4.5] for a complete discussion on the matter). A fundamental (theoretical) result claims that it is possible to perform any operator only with the set of one-qubit gates and one "sufficiently entangling" 2-qubit gate such as the CNOT [39]. To be able to implement any quantum algorithm with a given piece of hardware, it is therefore necessary to first find a universal set of technologically implementable gates. For instance the IBM quantum machine with superconducting qubits uses only special unitary gates on one qubit and the CNOT [40]. A technology using trapped ions will have other gates available like the MS gate [41,42]. Linear quantum optics will instead focus on CNOTs and one-qubit gates [43].

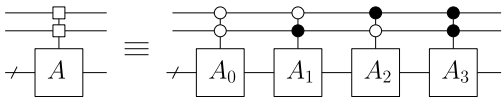

Fig. 2. Circuit equivalence for a multiplexor.

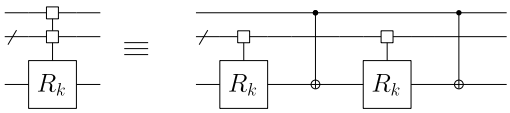

Fig. 3. Decomposition of a rotation multiplexor.

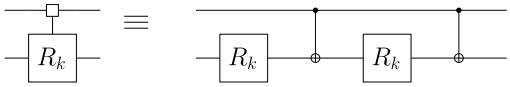

Fig. 4. Decomposition of a rotation multiplexor with one control qubit.

But having an implementable universal set of gates is not enough: if we are given a quantum operator as a unitary matrix, one also has to find a way to turn the desired unitary matrix acting on a potentially large number of qubits into a quantum circuit made of local, elementary gates. This problem is known as *circuit synthesis*, or equivalently, *compilation* of the unitary into a circuit. Note that the state preparation is a special case of circuit synthesis where we want to synthesize only the first column of a unitary matrix.

In this paper, we focus on the set consisting of the CNOT gate and all the one-qubit gates. The continuous aspect of the set makes it amenable to linear algebraic operations, and yet it can easily be mapped to other universal gate-sets [44–47], using for example the Solovay–Kitaev theorem [48] or more recent techniques [49–51].

Multiplexors. Some circuit structures are expressive enough to be reused in other constructions, thus helping during the compilation process. The most important of them is the multiplexor [23,52,53]. It can be regarded as a generalization of controlled gates because it applies a different operator for each value of the control qubits. For instance a multiplexor controlled by two qubits and acting on k qubits has the matrix block structure

$$A = \begin{pmatrix} A_0 & 0 & 0 & 0 \\ 0 & A_1 & 0 & 0 \\ 0 & 0 & A_2 & 0 \\ 0 & 0 & 0 & A_3 \end{pmatrix}$$

where A_0 is a k-qubit operator that is applied if both control qubits are 0, A_1 is applied if the first control qubit is 0 and the second one is 1, *etc.* The graphical representation of a multiplexor is exemplified in Fig. 2 with the correspondence between A and a succession of multi-controlled gates. In the circuit, the crossed-out line stands for a several qubits (in this case, k qubits).

The problem of decomposing a multiplexor into elementary gates admits algorithms with varying numerical costs depending on the choice of elementary gates [23,53]. In the case of a multiplexor applying only one kind of elementary rotations (e.g. along on of the axis X, Y or Z – we call such a structure a rotation multiplexor) the transition from the angles of the multiplexors to the angles in the quantum circuits can be done via a single matrix/vector product [53]. Moreover, the decomposition is much simpler than the general case shown in Fig. 2: the decomposition of a R_k-multiplexor controlled by n qubits into two multiplexors controlled by $n-1$ qubits has the shape shown in Fig. 3 and the special case with one control qubit is shown in Fig. 4 (where we omit angles for legibility). Such decompositions can be applied recursively and by removing some CNOT gates that cancel (see Figure 2 in [23] for more details) we obtain a final quantum circuit composed of 2^{n-1} CNOTs and 2^{n-1} elementary rotations. For multiplexors in $SU(2)$ the decomposition given in Fig. 3 remains valid up to a diagonal matrix that replaces the last CNOT gate. Hence, without considering this extra gate – for our purpose we will be able to remove it – we need $2^{n-1}-1$ CNOTs and 2^{n-1} generic one-qubit gates to implement an $SU(2)$-multiplexor.

Quantum state preparation. A common method for preparing a generic quantum state on n qubits consists in applying a series of operations such that we are left with the preparation of a quantum state on $n-1$ qubits, and we repeat the process until we have to prepare only a one-qubit state. Desentangling the first qubit is for instance equivalent to zeroing the second half of the components of the corresponding vector Ψ. To do so, one can apply for each bitstring $s \in F_2^{n-1}$ a specific two-qubit operation U_s on the first qubit such that $U_s(\Psi_{0s}|0s\rangle + \Psi_{1s}|1s\rangle) = \Psi'_{0s}|0s\rangle$. Then the global operator $\bigoplus_s U_s$ can be implemented either by applying successively one R_z-multiplexor and one R_y-multiplexor, both on the first qubit and controlled by the $n-1$ other ones, or by applying one $SU(2)$-multiplexor, still on the first qubit and controlled by the other qubits [21,23]. In the case where we only use R_y and R_z multiplexors, we can simply repeat the operation on the $n-1$ remaining qubits. Overall we need to implement two rotation multiplexors on n qubits, two on $n-1$ qubits, etc. for a total of $2 \times \sum_{k=2}^{n} 2^{k-1} \approx 2^{n+1}$ CNOTs and $2 + 2 \times \sum_{k=2}^{n} 2^{k-1} \approx 2^{n+1}$ elementary rotations. Some optimizations can decrease the CNOT-count by a linear term in n but we focus on the asymptotic complexity. When using multiplexors in $SU(2)$, we remark that the additional diagonal gate in the synthesis of the multiplexor can be merged with the remaining quantum state as adding phases to each component of the state will not change the number of nonzero elements. So preparing a quantum state with multiplexors in $SU(2)$ requires to implement one $SU(2)$-multiplexor on n qubits, one on $n-1$ qubits, etc. (without considering the extra diagonal gates) for a total of approximately 2^n CNOTs and 2^n generic one-qubit gates. Finally, to have the total count for the number of elementary rotations, we decompose each one-qubit gate U as a product of three elementary rotations (ignoring the global phase) [38]

$$U = R_x(\alpha) \times R_z(\beta) \times R_x(\gamma) \tag{1}$$

where α, β, γ are three real parameters. R_x rotations commute with the CNOT gate if the R_x gate acts on the target qubit of the CNOT gate. So for each quantum subcircuit implementing an $SU(2)$-multiplexor and starting from the leftmost rotation, we can commute the R_x gate, merge it with the next generic one-qubit gate, and repeat the process (decomposition shown in Eq. (1), commutation and merging) until we reach the last one-qubit gate of the multiplexor implementation. Thus, up to a linear number of gates, all the generic one-qubit gates can be decomposed into only two elementary rotations, for a total of approximately 2^{n+1} rotation gates.

Quantum Shannon decomposition. Among the various existing synthesis methods [38,54,55], the one giving the shortest circuits in terms of number of gates is the Quantum Shannon Decomposition (QSD) [21,23]. It relies on the following two decomposition formulas:

- the first one is the Cosine–Sine decomposition (CSD) of a unitary matrix on n qubits U [31]:

$$U = \begin{pmatrix} A_1 & \\ & A_2 \end{pmatrix} \begin{pmatrix} C & -S \\ S & C \end{pmatrix} \begin{pmatrix} B_1 & \\ & B_2 \end{pmatrix}. \tag{2}$$

  A_1, A_2, B_1, B_2 are unitary matrices on $n-1$ qubits and C, S are real positive diagonal matrices such that $C^2 + S^2 = I_{2^{n-1}}$.

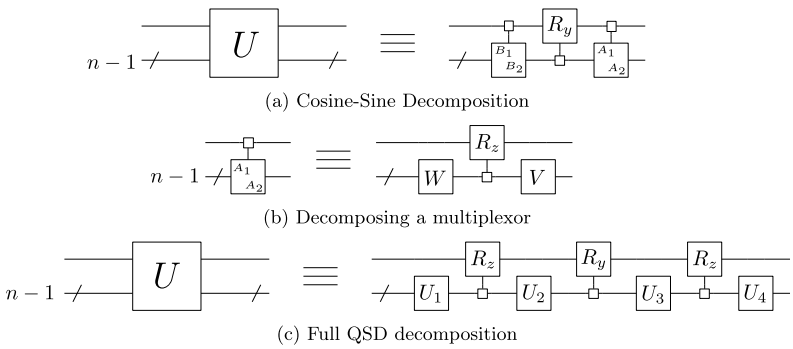

Fig. 5. Circuit equivalences for the QSD.

The second term in the CS decomposition is in fact a R_y-multiplexor controlled by the $n-1$ least significative qubits. The circuit equivalence is given in Fig. 5(a), where angles are omitted for legibility.

- The second formula decomposes a multiplexor

$$\begin{pmatrix} A_1 & \\ & A_2 \end{pmatrix} = \begin{pmatrix} V & \\ & V \end{pmatrix} \begin{pmatrix} D^\dagger & \\ & D \end{pmatrix} \begin{pmatrix} W & \\ & W \end{pmatrix} \tag{3}$$

with D a diagonal matrix on $n-1$ qubits, and V and W are unitary operating $n-1$ qubits. The second term involving the matrix D is in fact a R_z multiplexor controlled by the $n-1$ least significative qubits. The circuit equivalence is represented in Fig. 5(b), again with omitted angles.

Finally, synthesizing U on n qubits is equivalent to synthesizing 3 rotation multiplexors on n qubits and 4 matrices on $n-1$ qubits on which we can apply the QSD again as shown in Fig. 5(c). We repeat the process until we get only multiplexors and gates acting on a small number of qubits (typically 2) for which an exact decomposition is known [56–60].

If the Quantum Shannon Decomposition gives the best asymptotic number of CNOTs in the circuit: $\frac{23}{48} \times 4^n$, this method has nonetheless drawbacks. It does not take into account other metrics useful to minimize in the compilation process such as the classical time required to compute the circuit. The algorithm for computing Formula (2) consists in reducing the $K \times K$ matrix U into a 2×2 bidiagonal block form, then the 4 bidiagonal blocks are simultaneously diagonalized using bidiagonal SVD algorithms. The first part is the more expensive in terms of floating point operations: by applying Householder reflectors to the left and right of U, we progressively bidiagonalize U – this requires $K^3/3$ flops for each block – and we store the accumulation of each Householder reflector to compute A_1, A_2, B_1, B_2 – this requires $K^3/6$ flops for each block. Overall, computing the CSD on a $K \times K$ matrix requires $2 \times K^3$ flops [61]. Concerning Formula (3), one has to perform two matrix/matrix products and an eigenvalue decomposition. With square matrices of size K, each matrix/matrix product on matrices requires $2 \times K^3$ flops and the eigenvalue decomposition needs around $26 \times K^3$ [62, Table 3.13]. Overall for the first step of the Quantum Shannon Decomposition of a matrix of size N we have to compute one CSD of a matrix of size N and decompose two multiplexors, i.e. four matrix/matrix products of size $N/2$ and two eigenvalue decompositions of size $N/2$ too. This represents a total of $2\times N^3+4\times 2\times (N/2)^3+2\times 26\times (N/2)^3 = \frac{19}{2}\times N^3$ flops. Then to pursue the algorithm we have to perform the same operations on 4 matrices of size $N/2$, then on 16 matrices of size $N/4$, etc. Overall, with $N = 2^n$ we can approximate the total number of flops to 19×8^n which is very expensive. In the next section we propose an alternative method based on Householder transformations. Strongly connected to classical results about QR decomposition, this method aims at achieving better performance in the synthesis of quantum circuits by finding a compromise between circuit size and calculation time.

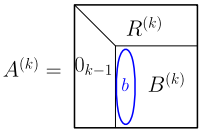

Fig. 6. Matrix pattern at step kth of Householder transformation.

3. Householder algorithm for unitary matrices

In this section, we first recall the main principles of the QR factorization of a general complex square matrix via Householder transformations. Then we consider the special case of unitary matrices that correspond to quantum operators.

The QR decomposition of a matrix $A \in \mathbb{C}^{n\times n}$ expresses A as the product of a unitary matrix $Q \in \mathcal{U}(n)$ and an upper triangular matrix R. A standard algorithm to compute such a factorization consists in applying a series of Householder transformations [31, p. 209] zeroing out successively the subdiagonal entries of each column.

At step k ($1 \le k \le n-1$) of the QR algorithm, we zero out all but the first entry of the vector b in the matrix depicted in Fig. 6 using the Householder transformation, $H'_k = I_n - \tau_k u_k u_k^\dagger$, where $u_k \in \mathbb{C}^{n-k+1}$ and $\tau_k = 2/u_k^\dagger u_k$. Note that in the complex case, the Householder matrix H'_k can be sometimes referred to as "elementary unitary matrix" (e.g. in [63]).

Then the kth iteration ends with the computation of the matrix

$$A^{(k)} = H_k A^{(k-1)},$$

with $H_k = \begin{pmatrix} I_{k-1} & 0 \\ 0 & H'_k \end{pmatrix}$, $A^{(0)} = A$ and $H_1 = H'_1$. This operation updates $B^{(k)} = A^{(k)}(k:n, k:n)$ via the relation

$$\left(I_{n-k+1} - \tau_k u_k u_k^\dagger\right) B^{(k)} = B^{(k)} - \tau_k u_k \left(\left(B^{(k)}\right)^\dagger u_k\right)^\dagger, \tag{4}$$

which zeros out the subdiagonal entries in column k (but does not affect the zeros already introduced in previous columns). The problem is now to find the vector $u_k \in \mathbb{C}^{n-k+1}$ such that

$$\left(I_{n-k+1} - \tau_k u_k u_k^\dagger\right) b = (\beta_k, 0, \ldots, 0)^T = \beta_k e_1.$$

with $\beta_k \in \mathbb{C}$. From [31, p. 233], we have $u_k = b \pm e^{i\theta}\|b\|e_1$ with $\theta = \arg(b_1)$ but various choices for u_k have been proposed in numerical libraries (see [63] for a review of these choices).

At the end of the algorithm we have computed a set of $n-1$ Householder transformations $H_1, H_2, \ldots, H_{n-1}$ such that

$$\left(\prod_{i=1}^{n-1} H_{n-i}\right) A = R$$

where R is upper triangular. Since the Householder matrices are Hermitian, we obtain

$$A = \left(\prod_{i=1}^{n-1} H_i\right) R = QR.$$

In the QR algorithm, the Householder matrices H'_k never need to be explicitly formed and the expensive part of the computation is the update of the matrix $B^{(k)}$, given in Eq. (4), which requires at each iteration a matrix–vector multiplication followed by a rank-1 update of $B^{(k)}$. The total cost of the factorization is about $\frac{4}{3}n^3$ complex flops ($\frac{16}{3}n^3$ real flops).

A block version of the algorithm uses the fact that a product of p Householder matrices $H_1 \times \ldots \times H_p$ can be written as $I - VTV^\dagger$ where V is an $n \times p$ rectangular matrix with the Householder vector $u_k \in \mathbb{C}^n$ at the kth column and T is an upper triangular matrix [64]. The algorithm consists in partitioning A into blocks of size $n \times n_A$ for some n_A, factorizing the first block and updating the remaining blocks via the operation (we use a matlab-like notation)

$$A(:, n_A + 1 : n) = A(:, n_A + 1 : n) - VTV^\dagger A(:, n_A + 1 : n),$$

and repeating the process with the next block until the whole matrix is triangularized. The update is richer in BLAS 3 operations [65], potentially leading to better performance, yet without decreasing the flop count [66].

Let now exploit the specificity of quantum operators where the corresponding matrix A is unitary and see how the QR decomposition simplifies. In this case, the triangular factor is also unitary and thus diagonal and the QR algorithm of A consists in a progressive diagonalization of A. For sake of simplification, we detail in the remainder only the first iteration. Let $b = (b_1, \ldots, b_n)^T$ be the first column of the unitary matrix A and $r = (r_1, \ldots, r_n)$ its first row. We choose the value of the Householder vector $u = b \pm e^{i\theta}\|b\|e_1$ as defined in [31, p. 233] but we will choose the sign "+". This choice has the advantage of maximizing $\|b\|$ (for sake of stability [31, p. 233]) and of simplifying the final decomposition of the quantum operator into elementary circuits, as we will see in Section 4. Since A (and $A^{(k)}$ at the kth iteration) is unitary, we have $\|b\| = 1$ and we get

$$u = b + e^{i\theta}e_1$$

and

$$\tau = \frac{2}{\|u\|^2} = \frac{2}{\|b\|^2 + \|e_1\|^2 + 2|b_1|} = \frac{1}{1 + |b_1|}.$$

Then applying the Householder transformation H to b gives

$$Hb = -e^{i\theta}\|b\|e_1 = -e^{i\theta}e_1. \tag{5}$$

The gain in complexity occurs in the update phase. Using the orthonormality of the vectors of A, the update expressed in Eq. (4) simplifies to

$$HA = A - \tau(b + e^{i\theta}e_1)\left(A^\dagger\left(b + e^{i\theta}e_1\right)\right)^\dagger = A - \tau(b + e^{i\theta}e_1)(e_1^T + e^{-i\theta}r).$$

Then we have

$$HA = A - \tau(be_1^T + e^{i\theta}e_1e_1^T + e_1r + e^{-i\theta}br).$$

The first column of A does not need to be updated in this computation because of Eq. (5) then we can ignore the term $be_1^T + e^{i\theta}e_1e_1^T$. Similarly the first row of A does not need to be updated because the unitarity of the rows of A ensures that $r = -e^{i\theta}e_1^T$ after application of H. Moreover $\tau e^{-i\theta} = 1/(e^{i\theta} + |b_1|e^{i\theta}) = 1/(b_1 + e^{i\theta}) = 1/u_1$, where u_1 denotes the first component of u. So we are left with the rank-1 update

$$(HA)_{2:n,2:n} = A_{2:n,2:n} - \frac{b(2:n) \cdot r(2:n)}{u_1}.$$

The matrix–vector product expressed in Eq. (4) for the classical QR factorization is avoided. The matrix obtained after the first iteration is then

$$A^{(1)} = \begin{pmatrix} -e^{i\theta} & 0 \\ 0 & (HA)_{2:n,2:n} \end{pmatrix}$$

and we can continue the algorithm on the unitary matrix $A^{(1)}(2 : n, 2 : n)$ and so on, until A becomes diagonal. The update at the kth iteration requires only $(n-k)^2$ multiplications and $(n-k)^2$ additions. Finally this new algorithm requires $\sum_{k=1}^{n-1} 2\times(n-k)^2 \sim \frac{2}{3}n^3$ complex flops, which is twice as less than the standard case.

It is possible to choose the vector u such that $u_1 = 1$, then the value of τ will be adjusted so that the resulting Householder transformation H remains the same. More precisely, keeping the notations above we set

$$u \leftarrow \frac{1}{e^{i\theta}(1 + |b_1|)}u \tag{6}$$

and we obtain $\tau = (1 + |b_1|)$ and then the update phase becomes

$$(HA)_{2:n,2:n} = A_{2:n,2:n} - u(2:n) \cdot r(2:n). \tag{7}$$

The algorithm can easily be done in place. One can store the Householder vectors in the strictly lower triangular part of A, the diagonal elements of R are stored in the diagonal and the τ_i's are stored in a specific array.

The main cost of the algorithm resides in the rank-one update phase in Eq. (7). In order to use more optimized BLAS 2 and BLAS 3 operations we can derive from Eq. (7) new update relations. Suppose we have already performed the factorization and the update for the first nb rows and columns for some nb. Therefore the first nb columns of A contain the Householder vectors, and the block $A(1 : nb, nb + 1 : n)$ has been updated following (7). Let $i, j \in [\![nb + 1, n]\!]$, one can verify that the update of the element $A(i, j)$ is given by

$$A(i, j) \leftarrow A(i, j) - \sum_{k=1}^{nb} A(i, k) \times A(k, j) \tag{8}$$

by simply applying successively the update (7).

In terms of matrix and vector operations we have

$$A(i, nb+1 : n) \leftarrow A(i, nb+1 : n) - A(i, 1 : nb) \times A(1 : nb, nb+1 : n) \tag{9}$$

for the update of one row,

$$A(nb+1 : n, j) \leftarrow A(nb+1 : n, j) - A(nb+1 : n, 1 : nb) \times A(1 : nb, j) \tag{10}$$

for the update of one column and

$$\begin{aligned} A(nb + 1 : n, nb + 1 : n) \leftarrow \\ A(nb + 1 : n, nb + 1 : n) - A(nb + 1 : n, 1 : nb) \\ \times A(1 : nb, nb + 1 : n) \end{aligned} \tag{11}$$

for the update of the full matrix. This last update is a BLAS 3 operation and can potentially yield higher performance on hybrid CPU–GPU architectures [67].

Using these new update relations we can improve the algorithm by three means:

- first we can improve the unblocked algorithm. Instead of updating the whole matrix at each iteration with a rank one update we only update one row and one column: at the kth iteration we have computed the kth Householder vector and we update the row $A(k + 1, k + 1 : n)$ and the column $A(k+1 : n, k+1)$ via the relations (9) and (10). Such updates consist in more and bigger matrix/vector operations and experimentally it appears to scale better.

- Secondly this naturally leads to a blocked version of the algorithm. Let nb be the size of our block. Once we have done the computations on the first nb rows and nb columns of A with an unblocked version, we can update the rest of the matrix with a matrix/matrix product via Eq. (11) and continue the algorithm on the matrix $A(nb+1:n, nb+1:n)$ until we reach the last block where the unblocked algorithm is applied.
- A third improvement can be made in order to avoid using the unblocked algorithm to compute the full panel of nb rows and columns of A. Indeed the update of the blocks $A(nb+1:n, 1:nb)$ and $A(1:nb, nb+1:n)$ can be performed with BLAS 3 operations. One can prove that there exist triangular matrices T_1^i, T_2^i of size $i \times i, i = 1..nb$ such that

$$A(i+1:n, 1:i) \leftarrow A(i+1:n, 1:i) \times T_1^i \tag{12}$$

$$A(1:i, i+1:n) \leftarrow T_2^i \times A(1:i, i+1:n). \tag{13}$$

The matrices T_1^i, T_2^i are computed using the following recursive formula:

$$T_1^1 = 1, T_1^{i+1} = \begin{pmatrix} T_1^i & -p_{i+1}T_1^i/\mathcal{N}_i \\ & 1/\mathcal{N}_i \end{pmatrix}, \tag{14}$$

$$T_2^1 = 1, T_2^{i+1} = \begin{pmatrix} T_2^i & \\ -q_{i+1}T_2^i & 1 \end{pmatrix} \tag{15}$$

with $p_i = A(1:i, i)$, $q_i = A(i, 1:i)$. $\mathcal{N}_i$ is the normalization factor expressed in Eq. (6).

Proof of Formulas (12)to(15). By induction on i. We do it for T_2 only. The case $i = 1$ is trivial because we do not have to update the first row. Now suppose the result is true for some i, $1 \leq i < nb$. The first i rows are already updated by the application of T_2^i, we only need to update the next row $i+1$. Let $A(i+1, j), j \in [\![nb+1, n]\!]$ be an element of this row. If the column $A(1:i, j)$ was already updated the update of $A(i+1, j)$ would be given by Eq. (8), i.e.

$$A(i+1, j) \leftarrow A(i+1, j) - \sum_{k=1}^{i} A(i+1, k) \times A(k, j).$$

Written differently $A(i+1, j) \leftarrow A(i+1, j) - A(i+1, 1:i+1) \cdot A(1:i, j)$. By hypothesis $A(1:i, j)$ is updated by the relation $A(1:i, j) \leftarrow T_2^i A(1:i, j)$. This gives

$$A(i+1, j) \leftarrow [-A(i+1, 1:i+1) \times T_2^i \; ; \; 1] \cdot A(1:i+1, j).$$

Doing it for all j and concatenating it with the update of the first i rows by the action of T_2^i gives the result. The same thing can be done with T_1 but one has to be careful about the normalization of the Householder vectors. □

Therefore T_1^{nb} and T_2^{nb} only depend on the block $A(1:nb, 1:nb)$ and can be used to update $A(nb+1:n, 1:nb)$ and $A(1:nb, nb+1:n)$ in two BLAS 3 updates. This means that during an iteration we only need to perform an unblocked Householder factorization on a square matrix of size nb and then perform 3 BLAS 3 updates. The pseudo code of the algorithm is given in Algorithm 1 (we call the corresponding routine ZUNQRF and its unblocked version ZUNQR2). ZLARFT2 refers to the adaptation of the standard ZLARFT routine that computes the triangular matrices.

Thanks to the above QR decomposition resulting in a product of Householder matrices and a diagonal matrix, we store the information of a unitary matrix into the subdiagonal part of the complex matrix (the Householder vectors), and two real vectors containing the θ's (angles of the diagonal entries) and the τ's. In the next section we use this factorization of unitary operators to obtain quantum circuits.

Algorithm 1 Householder factorization of a unitary matrix A - ZUNQRF

Require: $N \geq 0$, $A \in \mathcal{U}_N$
Ensure: $A = QR$
// NX determines when to switch from blocked to unblocked code
// NB is the block size
for $I = 1, NX, NB$ **do**
  $IB \leftarrow MIN(N - I + 1, NB)$
  call ZUNQR2(IB, IB, A(I, I), TAU(I))
  $T_1, T_2 \leftarrow$ ZLARFT2(N, IB, A(I, I), TAU(I))
  update $A(I:N, I:I+IB)$ via a call to ZTRMM
  if $I + IB \leq N$ **then**
    update $A(I:I+IB, I:N)$ via a call to ZTRMM
    update $A(I+IB:N, I+IB:N)$ via a call to ZGEMM
  end if
end for
if $I \leq N$ **then**
  call ZUNQR2(N-I+1, N-I+1, A(I, I), TAU(I))
end if

4. From the Householder decomposition to a quantum circuit

In this section we develop several methods to convert the Householder representation of a unitary matrix into a quantum circuit. We present a general method in Section 4.1 and we optimize it in Section 4.2.

4.1. General method

Let $U \in U(2^n)$ be the unitary matrix we want to synthesize. The QR factorization of U gives normalized vectors $u_1, u_2, \ldots, u_{2^n-1}$ and a diagonal matrix D such that

$$U = \prod_{i=1}^{2^n-1} H_i \times D.$$

where H_i are Householder matrices defined by $H_i = I_{2^n} - 2u_i u_i^\dagger$ as in Section 3 (since the u_i are normalized). The synthesis of a diagonal operator is a well-known problem [24,68]. Therefore, the main issue is the synthesis of the Householder matrices. We recall that

$$H_i u_i = -u_i$$

and

$$\forall v, v \perp u_i \Rightarrow H_i v = v.$$

Consequently, for any unitary matrix P_i whose first column is u_i we can write

$$H_i = P_i D_G P_i^\dagger \tag{16}$$

with

$$D_G = \begin{pmatrix} -1 & & & \\ & 1 & & \\ & & \ddots & \\ & & & 1 \end{pmatrix}.$$

Indeed, P_i can be regarded as an orthonormal basis of vector columns containing u_i. In other words, Eq. (16) is a diagonalization of H_i.

From this analysis, we get the following decomposition of the unitary matrix U

$$U = \prod_{i=1}^{2^n-1} P_i D_G P_i^\dagger \times D$$

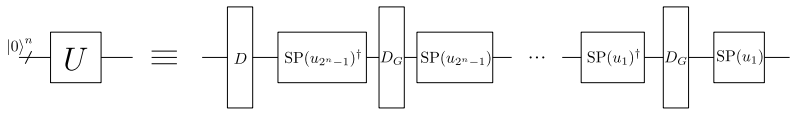

Fig. 7. A first circuit for the Householder method.

and we can derive a quantum circuit as depicted in Fig. 7.

- As mentioned already, the synthesis of D is a problem with known solutions.
- Each block $P_i D_G P_i^\dagger$ is equivalent to de-preparing the state u_i, applying D_G and re-preparing the state u_i.
  - The matrix D_G is the "zero phase shift" operator and is used for instance in the Grover diffusion operator in Grover's algorithm [3].
  - In our circuits we use the notation $SP(v)$ to refer to a black box who prepares the state v. Although many different operators can prepare the state v we insist on the fact that in one circuit the operators preparing and de-preparing the same state are exactly the same, otherwise the decomposition would not be valid. Many previous research studies have sought to optimize the preparation of states and we use their results for our synthesis [21,23,69].

4.2. Resources estimation

We now turn to the question of the size of the circuit sketched in Fig. 7 (measured in number of CNOTs), and to the computational cost to generate the circuit (measured in flops). In this section we only give asymptotic results which are summarized in Tables 2 and 3.

Asymptotically, it turns out that the synthesis of D and D_G are negligible. Using existing methods one can synthesize D in $O(2^n)$ gates, while the series of $2^n - 1$ subcircuits D_G requires at most $O((2^n - 1)n^2)$ gates [70] (when n is the number of qubits). As we will see in the following, the synthesis of the P_i's requires $O(4^n)$ gates: this is the dominant factor.

The complexity of the size of the circuit is therefore essentially due to the preparation and de-preparation of quantum states. Because of the structure of the problem, we can do better than systematically applying state preparation on n qubits for each of the u_i vectors. We describe two successive optimizations. The first one relies on the possibility to perform state preparations on less than n qubits; the second one proposes to fuse adjacent sequences of de-preparations and preparation of states.

4.2.1. Optimization based on state preparation

When preparing u_i, if only the last 2^k elements of $u_i \in \mathbb{C}^{2^n}$ are nonzero, the state is encodable on k qubits only. This means that a k-qubit operator can prepare the state $u_i' \in \mathbb{C}^{2^k}$ such that $u_i = \binom{0}{u_i'}$. Let Q be such an operator, then the operator

$$P = X^{\otimes(n-k)} \otimes Q = \begin{pmatrix} (0) & & 1 \\ & \iddots & \\ 1 & & (0) \end{pmatrix} \otimes \begin{pmatrix} u_i' & (*) \end{pmatrix} = \begin{pmatrix} 0 & * \\ u_i' & * \end{pmatrix}$$

prepares u_i. A quantum circuit to illustrate this is given in Fig. 8.

Thus, up to the operators X, D and D_G, we observe that the Householder method breaks down as follows: the synthesis of the first 2^{n-1} columns is done via operators acting on n qubits, then the next 2^{n-2} columns are synthesized with operators acting on $n-1$ qubits, *etc.*

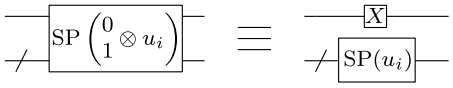

Fig. 8. Desentangling one qubit in state preparation.

In [70] the concept of isometries is formalized. Formally, with $n > m$, an m to n qubits isometry can be represented as a $2^n \times 2^m$ matrix V such that

$$V^\dagger V = I_{2^m \times 2^m}.$$

The data of the 2^m first columns of an operator on n qubits can be regarded as such an isometry V. For instance an isometry from 0 to n qubits is a quantum state on n qubits. Synthesizing an isometry from $n-1$ to n qubits is equivalent to synthesizing the first 2^{n-1} columns of an n-qubit operator. With this formalism the synthesis of an n-qubit operator via the Householder method naturally leads to synthesizing n isometries, more precisely, isometries from $k-1$ to k qubits for k from 1 to n. Therefore we introduce the following notations:

- h_k refers to the number of CNOTs necessary to the synthesis of an isometry from $k-1$ to k qubits with the Householder method,
- c_n refers to the number of CNOTs necessary to the synthesis of an n-qubit operator with the Householder method,

and, referring to the discussion above, we have

$$c_n \sim \sum_{k=1}^{n} h_k. \tag{17}$$

With this decomposition we voluntarily omit the side-effects that may occur between two subcircuits acting on a different number of qubits – typically the transition between the subcircuit preparing states on j qubits only and the subcircuit preparing states on $j+1$ qubits. These side-effects are asymptotically negligible and not taking them into account highly simplifies the calculations.

We are concerned with estimating c_n: to this end we focus on the estimation of h_k. A lower bound to the asymptotic behavior of h_k is given in [70]:

$$h_k^{min} = \frac{3}{16} 4^k + o(4^k).$$

With Eq. (17) we derive the lower bound $\frac{1}{4}4^n$ for c_n.

With our current circuit, we have

$$h_k \sim 2 \times (2^{k-1}) \times p_k \tag{18}$$

where p_k is the number of CNOTs required to prepare a state on k qubits. The value p_k varies depending on the structure of subcircuits we consider [21]:

- with rotation multiplexors, $p_k = 2^{k+1}$, $h_k \sim 2 \times 4^k$, hence $c_n \sim \frac{8}{3}4^n$;
- with multiplexors in $SU(2)$, $p_k = 2^k$, $h_k \sim 4^k$ and $c_n \sim \frac{4}{3}4^n$.

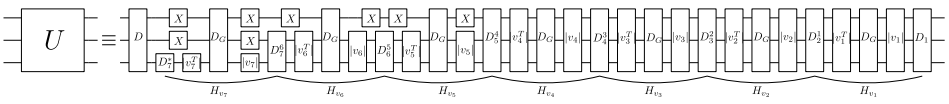

Fig. 9. Quantum circuit designed by the Householder method for 3 qubits.

Table 2
Asymptotic gate counts for decomposition methods.

Method	CNOT count	Rotation count
QR	8.7×4^n	Unavailable
Quantum Shannon	$23/48 \times 4^n$	$9/8 \times 4^n$
Householder (with rotation multiplexors)	2×4^n	2×4^n
Householder (with multiplexors in $SU(2)$)	4^n	2×4^n
Lower bound	$1/4 \times 4^n$	4^n

Table 3
Asymptotic flop counts for decomposition methods.

Method	Flops
Quantum Shannon	19×8^n
Householder	$2/3 \times 8^n$
Classical QR factorization	$4/3 \times 8^n$

The same calculation can be done for the number of rotations in the circuit. Actually, Eqs. (17) and (18) highlight the decomposition of the quantum circuit into smaller subcircuits, and thus remain true by replacing the number of CNOTs with the number of elementary rotations. A quantum state preparation on n qubits requires 2^{n+1} rotations, whether using rotations or $SU(2)$ multiplexors [21]. Overall the number of rotations r_n required for the synthesis of a n-qubit operator with the Householder method is

$$r_n = \frac{8}{3}4^n.$$

4.2.2. Optimizing adjacent state preparations and de-preparations

We now focus on the concatenations of the adjacent subcircuits $SP(u_{i+1})$ and $SP(u_i)^\dagger$ in the circuit of Fig. 7. These sequences of operations can indeed be optimized.

To this end we need to look into more details to the state preparation circuits. A circuit P that prepares the state ψ on n qubits can be decomposed as

$$P = DY$$

where

$$D = \begin{pmatrix} e^{i\theta_1} & & \\ & \ddots & \\ & & e^{i\theta_{2^n}} \end{pmatrix}$$

such that $\theta_j = arg(\psi(j))$ and Y prepares the real state

$$\Psi = \begin{pmatrix} |\psi_1| \\ \vdots \\ |\psi_{2^n}| \end{pmatrix}.$$

Y can be synthesized with only R_y rotations by following the standard methodology for state preparation [23] without caring about the phases equal to 0. Using this decomposition the products preparation/de-preparation that we encounter in the global decomposition are of the form

$$P_j^\dagger P_i = Y_j^T D_j^* D_i Y_i$$

and the diagonal matrices can merge, thus diminishing the size by a cost of a diagonal matrix. In total $2^{k-1} - 1$ diagonals on k qubits vanish. Asymptotically this represents a gain of $2/3 \times 4^n$ CNOTs and $2/3 \times 4^n$ rotations if we use rotation multiplexors, bringing the total to 2×4^n CNOTs and 2×4^n rotations. If we use multiplexors in $SU(2)$, only multiplexors on k qubits are merging and not diagonal anymore, saving twice less CNOTs but the same number of rotations. We save in total $1/3 \times 4^n$ CNOTs and $2/3 \times 4^n$ rotations and the number of CNOTs, resp. rotations, becomes asymptotically equal to 4^n, resp. 2×4^n. We also notice that the operators X that appear when we switch to synthesis on a lower number of qubits disappear too by multiplying themselves. An example on 3 qubits is shown in Fig. 9. We use the following notation: $|v_k|$ (resp. $|v_k^T|$) represents the operator that prepares (resp. deprepares) the real state $|v_k|$ consisting of the amplitudes of the components of the state v_k. D_k is the diagonal gate containing the phases of the components of the state v_k and $D_k^j = D_j^* \times D_k$. The results for the final gate counts are given in Table 2.

4.2.3. Flop counts

Apart from the circuit size, the other measure we are interested in is the computational cost, measured in flops.

The computational cost of the synthesis part is negligible compared to the cost of the Householder decomposition. Overall state preparations of states of size $2^n, 2^n - 1, \ldots, 3, 2$ need to be performed. For a state on k qubits, it requires $O(k2^k)$ operations, and we need to do it for 2^{k-1} states. Thus the synthesis part needs around

$$\sum_{k=1}^{n} k2^k \times 2^{k-1} = O(n4^n)$$

floating point operations. This is asymptotically negligible compared to the Householder factorization where $O(8^n)$ operations are needed. Table 3 summarizes the flop count for the various methods.

5. Experimental results

The experiments have been carried out on one node of the QLM (Quantum Learning Machine) located at ATOS/BULL. This node is a 24-core Intel Xeon(R) E7-8890 v4 processor at 2.4 GHz. Hyper-threading has been disabled.

Most of the programs are written in C with the C-interface for LAPACK [71] (LAPACKE). We adapted the LAPACK routine ZGEQRF (in Fortran) to compute the QR factorization of unitary matrices using the blocked algorithm described in Section 3. LAPACK is linked with the MKL [72] multithreaded BLAS. The original ZGEQRF routine computes the best block size according to the size of the matrix and the hardware, we keep this computation in our modified routine. Our experiments use random unitary matrices generated via the LAPACK routine ZLAROR which generate matrices from a uniform distribution according to the Haar measure [73]. This way we get the most generic matrices possible: dense, without any particular structure or pattern in the matrix elements. We are thus ensured to have a worst case scenario in terms of performance for our algorithms.

We present here numerical experiments to evaluate successively the sequential performance, the strong scalability and the weak scalability using multiple cores.

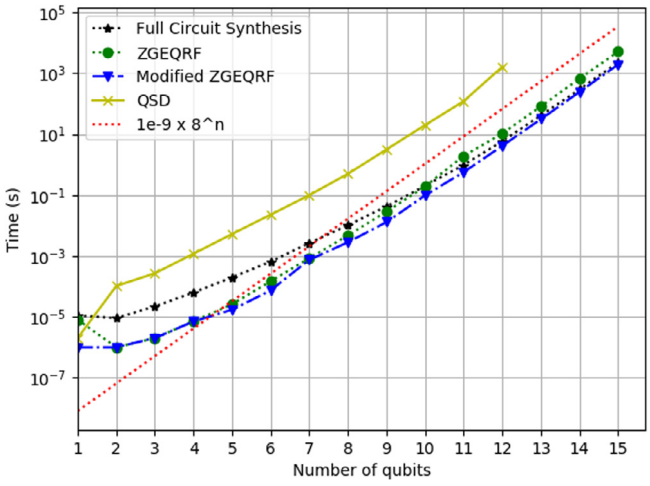

Fig. 10. Sequential time for operator decomposition and circuit synthesis.

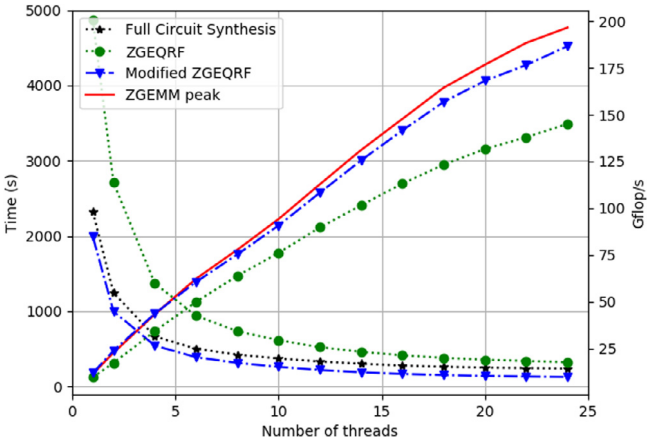

Fig. 11. Strong scaling for quantum operator decomposition and circuit synthesis on 15 qubits.

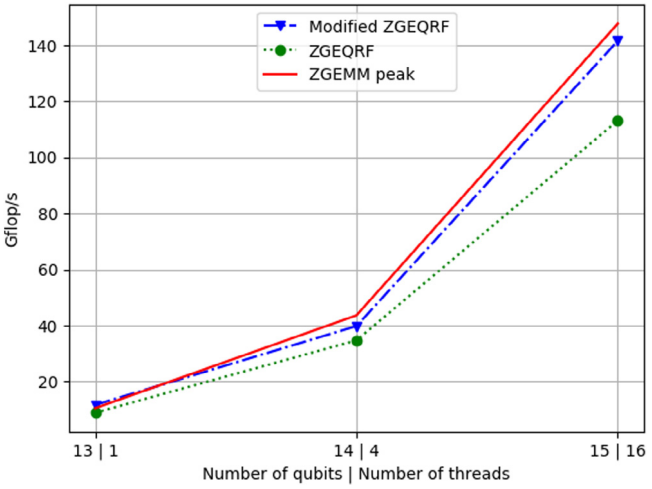

Fig. 12. Weak scaling for quantum operator decomposition on 15 qubits.

5.1. Sequential runs

In Fig. 10 we compare the performance (in time) of the following routines or programs:

- The LAPACK routine ZGEQRF that computes the QR factorization of a complex matrix in double precision (note that here the matrix is square), to serve as a reference.
- Our modified ZGEQRF routine adapted for unitary matrices.
- The complete circuit synthesis process which includes the QR factorization and the synthesis of the circuit obtained from this decomposition as explained in Section 4.
- The Quantum Shannon Decomposition (QSD), where the implementation essentially relies on the methodology described in [23] and uses the LAPACK routine ZUNCSD to compute the Cosine–Sine Decomposition (CSD). The routine implements the algorithm in [61]. This algorithm is the state of the art and has already been used in other implementations [74,75].

We considered matrices of sizes $2^k \times 2^k$, $k = 1 \ldots 15$ (operators acting on 1 to 15 qubits). The upper limit of 15 was chosen so that all decompositions can be achieved within an hour. This is why the curve plotting the QSD decomposition stops for 12 qubits.

As expected all the methods follow asymptotically $O(8^n)$ (curve also plotted) in accordance with the theoretical complexity. The gap between the general and modified QR factorizations in log scale corresponds approximatively to a factor of 2, in accordance with the flop count.

When comparing the QR and QSD methods, we observe that for the same amount of time we can synthesize matrices with 2, almost 3 qubits more. The ratio between the times taken by the QSD and our method is even increasing with the number of qubits, reaching a value of almost 300 for 12 qubits which is much bigger than the expected ratio of 30. This is due to the routine ZUNCSD that does not follow the theoretical complexity and does not scale well with the number of qubits.

5.2. Multithreaded runs

Because we could reach 15 qubits (unitary matrices of size $32\,768 \times 32\,768$) with a sequential run in less than one hour, we chose this size for our multithreaded runs. The strong scalability is then evaluated using up to 24 threads. Since the ZUNCSD routine used for the QSD is not parallel, it has been excluded from our experiments. Fig. 11 presents performance results (in time and Gflop/s) for the chosen number of threads. The time of the modified ZGEQRF scales like the full circuit synthesis since the QR factorization represents most of the computational cost in the synthesis. Also, due to a smaller flop count, the modified QR is always much faster than the general QR. Moreover, looking at the Gflop/s performance rate, we observe that our modified QR factorization offers a good scalability due to an algorithm which is rich in BLAS 3 operations and provides a performance close to that of a matrix–matrix product (ZGEMM routine, also plotted in Fig. 11). Note that the Gflop/s rate for the full circuit synthesis is not plotted since the bulk of the arithmetical operations correspond to those of the factorization and the time of the synthesis itself is negligible.

Our experiments on weak scaling aim at measuring how the performance evolves with the number of threads but with a fixed problem size for each thread. Our algorithm for circuit synthesis can only accept matrices of size $2^n \times 2^n$, i.e. 4^n entries. As we can only multiply the size of our problems by a factor of 4, we need to multiply also the number of threads by a factor of 4. Thus, starting from a sequential run on 13 qubits, we achieved experiments on 14 and 15 qubits using 4 and 16 threads, respectively. The results given in Fig. 12 show that the rate (in Gflop/s) of the modified ZGEQRF increases with the number of threads/qubits with a very good scalability (close to that of ZGEMM) due to the mostly BLAS 3 operations implemented in the algorithm.

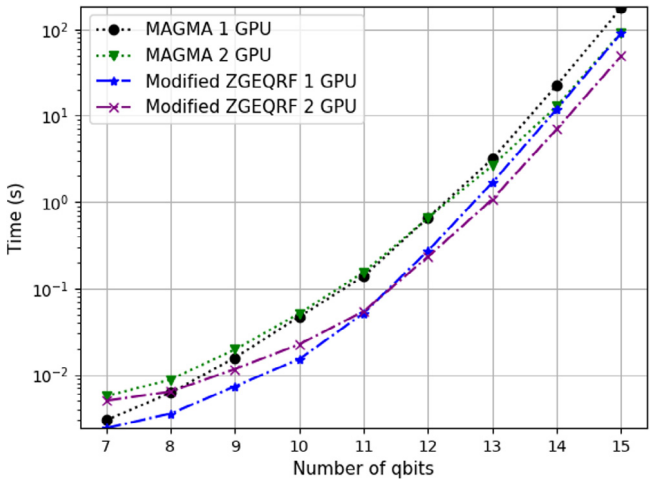

Fig. 13. Time for factorization of unitary matrices on GPUs.

5.3. Experiments on Graphics Processing Units (GPU)

We performed additional experiments to study the behavior of our QR algorithm for unitary matrices using two Kepler K40 with 2880 CUDA cores and a multicore host composed of two Intel Xeon E5-2620 processors (6 cores each). The time for the synthesis is not plotted here since it is negligible compared to the time of the QR factorization.

Similarly to what was made previously with the LAPACK routine, we modified the QR routine from the MAGMA [67] linear algebra library for GPUs according to Algorithm 3.1. Note that the transfer of the panel (block column factorized at each iteration) from the CPU to the GPU performed in MAGMA is replaced by a transfer of the 2 triangular matrices mentioned in Section 3 which are broadcasted to the GPUs involved in the computation. In Fig. 13, we obtain the factor of 2 (due to twice less flops) between the standard and the modified QR factorization. We also observe that using 2 GPUs has no interest for problems smaller than 12 qubits but we get a factor close to 2 (e.g., 1.84 for 15 qubits) when switching from 1 to 2 GPUs for problems larger than 13 qubits, showing a good scalability of the algorithm.

6. Conclusion

In this work we recalled the fundamentals of quantum computing and we stated the problem of quantum circuit synthesis. We highlighted the importance of having an efficient circuit synthesis framework by considering metrics based on flop and gate counts. To address this issue we presented a modified QR factorization in complex arithmetic based on Householder transformations where we exploit the specificities of unitary matrices to require twice as less flops and we proposed a scalable blocked implementation that contains mostly level-3 BLAS operations. Then we described a method to convert the QR factorization into a quantum circuit with clearly defined properties. Our method results in a significant gain in time compared to the best methods in quantum compiling. As future work, we will study the behavior of our method on bigger problems using large distributed HPC systems. In terms of circuit size, some improvements may be obtained by studying in more detail the optimization of state preparation occurring during the process.

Acknowledgments

This work was supported in part by the French National Research Agency (ANR) under the research project SoftQPRO ANR-17-CE25-0009-02, and by the DGE of the French Ministry of Industry under the research project PIA-GDN/QuantEx P163746-484124.

References

[1] R.P. Feynman, Int. J. Theor. Phys. 21 (1982) 467–488.
[2] D. Deutsch, Proc. Royal Soc. Lond. 400 (1985) 97–117, http://dx.doi.org/10.1098/rspa.1985.0070, URL: http://rspa.royalsocietypublishing.org/content/400/1818/97, arXiv:http://rsp.royalsocietypublishing.org/content/400/1818/97.full.pdf.
[3] L.K. Grover, Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing, ACM, New York, NY, USA, 1996, pp. 212–219.
[4] L. Chen, S. Jordan, Y.K. Liu, D. Moody, R. Peralta, R. Perlner, D. Smith-Tone, Report on Post-Quantum Cryptography, Technical Report NISTIR-8105, NIST – National Institute of Standards and Technology, 2016.
[5] P.W. Shor, SIAM Rev. 41 (1999) 303–332.
[6] J. Biamonte, P. Wittek, N. Pancotti, P. Rebentrost, N. Wiebe, S. Lloyd, Nature 549 (2017) 195.
[7] I. Kerenidis, A. Prakash, in: C.H. Papadimitriou (Ed.), 8th Innovations in Theoretical Computer Science Conference, ITCS'17, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017, pp. 49:1–49:21.
[8] I. Kerenidis, A. Prakash, Quantum gradient descent for linear systems and least squares, 2017, arXiv preprint arXiv:1704.04992.
[9] R. LaRose, A. Tikku, É. O'Neel-Judy, L. Cincio, P.J. Coles, npj Quantum Inf. 5 (2019) 8.
[10] A. Peruzzo, J. McClean, P. Shadbolt, M.H. Yung, X.Q. Zhou, P.J. Love, A. Aspuru-Guzik, J.L. O'brien, Nat. Commun. 5 (2014) 4213.
[11] A. Montanaro, Quantum walk speedup of backtracking algorithms, 2015, arXiv preprint arXiv:1509.02374.
[12] E. Farhi, J. Goldstone, S. Gutmann, A quantum approximate optimization algorithm, 2014, arXiv preprint arXiv:1411.4028.
[13] M. Suchara, Y. Alexeev, F. Chong, H. Finkel, H. Hoffmann, J. Larson, J. Osborn, G. Smith, Hybrid quantum–classical computing architectures, in: Proceedings of the 3rd International Workshop on Post-Moore Era Supercomputing, 2018, 2018.
[14] A.C.C. Yao, Foundations of Computer Science, 1993 Proceedings., 34th Annual Symposium on, IEEE, 1993, pp. 352–361.
[15] A. JavadiAbhari, S. Patil, D. Kudrow, J. Heckey, A. Lvov, F.T. Chong, M. Martonosi, Parallel Comput. 45 (2015) 2–17, http://dx.doi.org/10.1016/j.parco.2014.12.001.
[16] K.M. Svore, M. Troyer, IEEE Comput. 49 (2016) 21–030, http://dx.doi.org/10.1109/MC.2016.293.
[17] B. Valiron, N.J. Ross, P. Selinger, D.S. Alexander, J.M. Smith, Commun. ACM 58 (2015) 52–61, http://dx.doi.org/10.1145/2699415, URL: http://doi.acm.org/10.1145/2699415.
[18] E. Knill, Approximation by quantum circuits, 1995, arXiv preprint quant-ph/9508006.
[19] A. Barenco, C.H. Bennett, R. Cleve, D.P. DiVincenzo, N. Margolus, P. Shor, T. Sleator, J.A. Smolin, H. Weinfurter, Phys. Rev. A 52 (1995) 3457–3467.
[20] G. Cybenko, Comput. Sci. Eng. 3 (2001) 27–32, http://dx.doi.org/10.1109/5992.908999, URL: http://arxiv.org/abs/https://aip.scitation.org/doi/pdf/10.1109/5992.908999, arXiv:https://aip.scitation.org/doi/pdf/10.1109/5992.908999.
[21] M. Mottonen, J.J. Vartiainen, Decompositions of general quantum gates, 2005, arXiv preprint quant-ph/0504100.
[22] M. Reck, A. Zeilinger, H.J. Bernstein, P. Bertani, Phys. Rev. Lett. 73 (1994) 58–61, http://dx.doi.org/10.1103/PhysRevLett.73.58, URL: https://link.aps.org/doi/10.1103/PhysRevLett.73.58.
[23] V.V. Shende, S.S. Bullock, I.L. Markov, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 25 (2006) 1000–1010.
[24] S.S. Bullock, I.L. Markov, Quantum Inf. Comput 4 (2004) 27–47.
[25] J. Preskill, Quantum 2 (2018) 79.
[26] Y. Shi, N. Leung, P. Gokhale, Z. Rossi, D.I. Schuster, H. Hoffmann, F.T. Chong, Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems, ACM, 2019, pp. 103–1044.
[27] M. Amy, D. Maslov, M. Mosca, M. Roetteler, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 32 (2013) 818–830.
[28] L.E. Heyfron, E.T. Campbell, Quantum Sci. Technol. 4 (2019) 015004.
[29] O.D. Matteo, M. Mosca, Quantum Sci. Technol. 1 (2016) 015003.
[30] Y. Nam, N.J. Ross, Y. Su, A.M. Childs, D. Maslov, npj Quantum Inf. 4 (2018) 23, http://dx.doi.org/10.1038/s41534-018-0072-4, URL: https://doi.org/10.1038/s41534-018-0072-4.

[31] G.H. Golub, C.F. Van Loan, Matrix Computations, third ed., The Johns Hopkins University Press, Baltimore, 1996.
[32] R. Cabrera, T. Strohecker, H. Rabitz, J. Math. Phys. 51 (2010) 082101.
[33] P.A. Ivanov, E.S. Kyoseva, N.V. Vitanov, Phys. Rev. A 74 (2006) 022323, http://dx.doi.org/10.1103/PhysRevA.74.022323, URL: https://link.aps.org/doi/10.1103/PhysRevA.74.022323.
[34] J. Urías, D.A. Quiñones, Can. J. Phys. 94 (2016) 150–157, http://dx.doi.org/10.1139/cjp-2015-0490, URL: http://arxiv.org/abs/https://doi.org/10.1139/cjp-2015-0490, arXiv:https://doi.org/10.1139/cjp-2015-0490.
[35] C.F.V. Loan, J. Comput. Appl. Math. 123 (2000) 85–100, http://dx.doi.org/10.1016/S0377-0427(00)00393-9, URL: http://www.sciencedirect.com/science/article/pii/S0377042700003939. numerical Analysis 2000. III: Linear Algebra.
[36] R. Jozsa, N. Linden, Proc. R. Soc. Lond. 459 (2003) 2011–2032.
[37] G. Vidal, Phys. Rev. Lett. 91 (2003) 147902.
[38] M.A. Nielsen, I.L. Chuang, Quantum Computation and Quantum Information, Cambridge University Press, 2011.
[39] M.J. Bremner, C.M. Dawson, J.L. Dodd, A. Gilchrist, A.W. Harrow, D. Mortimer, M.A. Nielsen, T.J. Osborne, Phys. Rev. Lett. 89 (2002) 247902.
[40] D. Alsina, J.I. Latorre, Phys. Rev. A 94 (2016) 012314, http://dx.doi.org/10.1103/PhysRevA.94.012314.
[41] D. Maslov, New J. Phys. 19 (2017) 023035, URL: http://stacks.iop.org/1367-2630/19/i=2/a=023035.
[42] P. Schindler, D. Nigg, T. Monz, J.T. Barreiro, E. Martinez, S.X. Wang, S. Quint, M.F. Brandl, V. Nebendahl, C.F. Roos, M. Chwalla, M. Hennrich, R. Blatt, New J. Phys. 15 (2013) 123012, URL: http://stacks.iop.org/1367-2630/15/i=12/a=123012.
[43] P. Kok, W.J. Munro, K. Nemoto, T.C. Ralph, J.P. Dowling, G.J. Milburn, Rev. Modern Phys. 79 (2007) 135–174, http://dx.doi.org/10.1103/RevModPhys.79.135.
[44] A. Barenco, Proc. Royal Soc. Lond. A 449 (1995) 679–683.
[45] D. Deutsch, A. Barenco, A. Ekert, Proc. Royal Soc. Lond. 449 (1995) 669–677.
[46] D.P. DiVincenzo, Phys. Rev. A 51 (1995) 1015–1022, http://dx.doi.org/10.1103/PhysRevA.51.1015, URL: https://link.aps.org/doi/10.1103/PhysRevA.51.1015.
[47] S. Lloyd, Phys. Rev. Lett. 75 (1995) 346–349, http://dx.doi.org/10.1103/PhysRevLett.75.346, URL: https://link.aps.org/doi/10.1103/PhysRevLett.75.346.
[48] C.M. Dawson, M.A. Nielsen, Quantum Inf. Comput. 6 (2006) 81–95, URL: http://dl.acm.org/citation.cfm?id=2011679.2011685.
[49] V. Kliuchnikov, D. Maslov, M. Mosca, Quantum Inf. Comput. 13 (2013) 607–630.
[50] N.J. Ross, P. Selinger, Quantum Inf. Comput. 16 (2016) 901–953.
[51] P. Selinger, Quantum Inf. Comput. 15 (2015) 159–180.
[52] V. Bergholm, J.J. Vartiainen, M. Möttönen, M.M. Salomaa, Phys. Rev. A 71 (2005) 052330.
[53] M. Möttönen, J.J. Vartiainen, V. Bergholm, M.M. Salomaa, Phys. Rev. Lett. 93 (2004) 130502.
[54] A. De Vos, S. De Baerdemacker, Phys. Rev. A 94 (2016) 052317, http://dx.doi.org/10.1103/PhysRevA.94.052317, URL: https://link.aps.org/doi/10.1103/PhysRevA.94.052317.
[55] J.J. Vartiainen, M. Möttönen, M.M. Salomaa, Phys. Rev. Lett. 92 (2004) 177902.
[56] S.S. Bullock, I.L. Markov, Proceedings of the 40th Annual Design Automation Conference, ACM, New York, NY, USA, 2003, pp. 324–329.
[57] B. Kraus, J.I. Cirac, Phys. Rev. A 63 (2001) 062309, http://dx.doi.org/10.1103/PhysRevA.63.062309, URL: https://link.aps.org/doi/10.1103/PhysRevA.63.062309.
[58] F. Vatan, C. Williams, Phys. Rev. A 69 (2004) 032315, http://dx.doi.org/10.1103/PhysRevA.69.032315, URL: https://link.aps.org/doi/10.1103/PhysRevA.69.032315.
[59] F. Vatan, C.P. Williams, Realization of a general three-qubit quantum gate, 2004, arXiv preprint quant-ph/0401178.
[60] G. Vidal, C.M. Dawson, Phys. Rev. A 69 (2004) 010301, http://dx.doi.org/10.1103/PhysRevA.69.010301, URL: https://link.aps.org/doi/10.1103/PhysRevA.69.010301.
[61] B.D. Sutton, Numer. Algorithms 50 (2009) 33–65, http://dx.doi.org/10.1007/s11075-008-9215-6, URL: https://doi.org/10.1007/s11075-008-9215-6.
[62] S. Blackford, Benchmark lapack. http://www.netlib.org/lapack/lug/node71.html.
[63] R.B. Lehoucq, ACM Trans. Math. Software 22 (1996) 393–400, http://dx.doi.org/10.1145/235815.235817, URL: http://doi.acm.org/10.1145/235815.235817.
[64] X. Sun, C. Bischof, SIAM J. Matrix Anal. Appl. 16 (1995) 1184–1196.
[65] J. Dongarra, Int. J. High Perform. Comput. Appl. 16 (2002).
[66] R. Schreiber, C.V. Loan, SIAM J. Sci. Stat. Comput. 10 (1989) 53–57.
[67] S. Tomov, J. Dongarra, M. Baboulin, Parallel Comput. 36 (2010) 232–240.
[68] J.M. Welch, On the Synthesis of Quantum Circuits for Diagonal Operators in Quantum Computation (Doctoral dissertation), Harvard University, Graduate School of Arts & Sciences, 2015.
[69] M. Plesch, i.c.v. Brukner, Phys. Rev. A 83 (2011) 032302.
[70] R. Iten, R. Colbeck, I. Kukuljan, J. Home, M. Christandl, Phys. Rev. A 93 (2016) 032318, http://dx.doi.org/10.1103/PhysRevA.93.032318, URL: https://link.aps.org/doi/10.1103/PhysRevA.93.032318.
[71] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J.D. Croz, A. Greenbaum, S. Hammarling, A. McKenney, D. Sorensen, LAPACK Users' Guide, third ed., SIAM, Philadelphia, 1999.
[72] Intel, Math Kernel Library (MKL). http://www.intel.com/software/products/mkl/.
[73] G. Stewart, SIAM J. Numer. Anal. 17 (1980) 403–409.
[74] R. Iten, O. Reardon-Smith, L. Mondada, E. Redmond, R.S. Kohli, R. Colbeck, Introduction to universalqcompiler, 2019, arXiv preprint arXiv:1904.01072.
[75] J. Wang, K. Manouchehri, Physical Implementation of Quantum Walks, Springer, 2013.