DOI 10.34658/jacs.2020.28.1.39-49

JOURNAL OF APPLIED
COMPUTER SCIENCE
Vol. 28 No. 1 (2020), pp. 39-49

Loading Initial Data into the Quantum Register

Marcin Ostrowski[1]

[1]*Technical University of Łódź*
Institute of Information Technology FTIMS
ul. Wólczańska 215, 90-924 Łódź
marcin.ostrowski@p.lodz.pl

Abstract. *In this paper, we examine a simple algorithm for loading initial data into the quantum register. In order to perform the algorithm standard two input gates are used. The algorithm is tested for the Gaussian and sine wave states. In the Appendix full PyQuil code of the algorithm is attached.*
Keywords: *quantum computation, quantum simulation, quantum algorithms.*

1. Introduction

In the near future, quantum calculations can make a major contribution to the development of informatics [1]. Although practical implementations of quantum computer have not been built yet, its existence seems to be possible. Therefore, it is worth examining the properties of such machines.

Today we know Shor [2] and Grover [3] algorithms which are of lower computational complexity than their best classical counterparts. Another promising application of quantum computer are quantum simulations [4, 5, 6], i.e. the computer modeling of behavior of physical quantum systems. It gives the possibility of effective modeling quantum processes, which is not possible using classical computers [7]. Quantum computers can simulate a wide variety of quantum systems, including fermionic lattice models [8, 9], quantum chemistry [10, 11], and quantum field theories [12].

In previous works, we showed that a quantum computer can efficiently simulate processes such as free propagation of the Schrödinger particle [13] or diffusion of the particle inside an infinite potential well [14]. In the first case, we assumed that the initial state of the simulated particle (and thus the initial state of a quantum register) is a sampled Gaussian state given by Eq. (6). In the second work, the initial state of the particle takes the form of sine function in the form of Eq. (7). The process of introducing the initial state to the register has not been considered in any of these works. We discuss this issue here.

In this work we only consider Gaussian and sine wave states. However, the algorithm presented in this paper is universal. It can be used to enter any arbitrary state into the quantum register.

Issues similar to the one presented here have already been investigated in the literature. For example, the problem of finding the unitary operators performing the mapping between given initial and final states has been explored in [15]. The problem of decomposition of large unitary operators was presented in (e.g. [16]). The use of genetic algorithms as a possible solution to such problems was also discussed in [17, 18, 19]. Another group of problems connected with minimization of quantum automata can be found in [20].

The issue discussed here can be considered not only in the context of quantum simulation. The problem of loading initial data into the register appears in many other branches of quantum computing. For this reason, the algorithm presented here may be used in quantum signal processing [21, 22] as well as in quantum image processing [23].

2. Description of the algorithm

We assume that the initial state of the register (before executing the algorithm) is $|\psi_i\rangle = |000\ldots0\rangle$. The purpose of the algorithm is to enter into the register a final state in the form:

$$|\psi_f\rangle = z_0|0\ldots00\rangle + z_1|0\ldots01\rangle + z_2|0\ldots10\rangle + \cdots + z_{n_b-1}|1\ldots11\rangle, \qquad (1)$$

where parameters z_k are arbitrary complex values that satisfy the normalization condition ($\sum_{k=0}^{n_b-1} |z_k|^2 = 1$), $n_b = 2^{n_q}$ is number of base states of the register and n_q is number of qubits in the register.

The scheme of the algorithm is presented in Fig. 1. Gates denoted by Y are standard

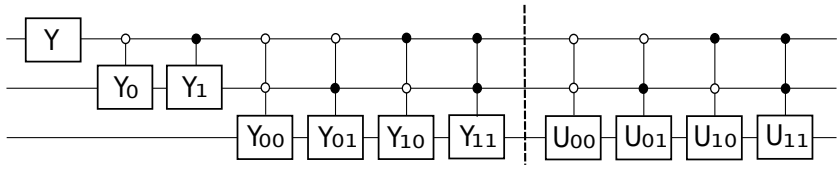

Figure 1: The scheme of the algorithm (example for $n_q = 3$)

R_y gates, which operates as follows:

$$Y = \begin{bmatrix} w_0 & w_1 \\ -w_1 & w_0 \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix}. \tag{2}$$

In the case of the one-qubit Y gate (first from the left in Fig. 1) coefficients w_i are given by:

$$w_0 = \sqrt{\sum_{k=0}^{\frac{n_b}{2}-1} |z_k|^2}, \qquad w_1 = \sqrt{\sum_{k=\frac{n_b}{2}}^{n_b-1} |z_k|^2}. \tag{3}$$

Coefficients w_i for Y gates with control qubits we can calculate using two dimensional array of real numbers in the form:

$$T_{i,j} = \frac{\sqrt{\sum_{k=0}^{q-1} |z_{k+jq}|^2}}{T_{0,j\backslash 2^i}\, T_{1,j\backslash 2^{i-1}} \,\ldots\, T_{i-1,j\backslash 2}}, \tag{4}$$

where $i = 0, 1, \ldots n_q - 1$ corresponds to number of control qubits, $j = 0, 1, \ldots p-1$, $p = 2^{i+1}$, $q = n_b/p$ and "\" is integer division. In the case of the first gate $w_0 = T_{0,0}$ and $w_1 = T_{0,1}$. In the case of the k-th gate with i control qubis $w_0 = T_{i,2k}$ and $w_1 = T_{i,2k+1}$.

In the second part of the algorithm (implementing phase part of the algorithm and separated by the dotted line in Fig. 1) we use gates marked as U_k which carry out the operations described as follows:

$$U_k = \begin{bmatrix} \exp(i\arg(z_{2k})) & 0 \\ 0 & \exp(i\arg(z_{2k+1})) \end{bmatrix}. \tag{5}$$

U_k gates can be implemented using standard phase-shift and NOT gates, as is shown in Fig. 2.

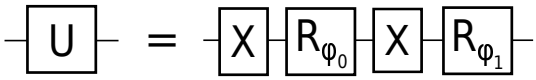

Figure 2: Realisation of U gates

Multi-input unitary gates U and Y (with multiple control qubits) are realized according to the scheme showed in Fig. 3. (Which is well-known and presented in the literature, e.g. [24].)

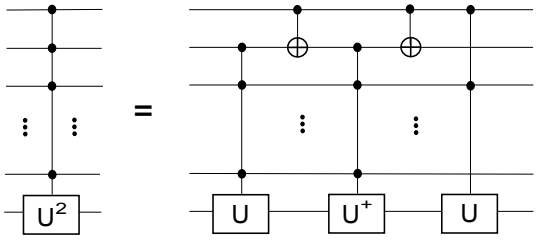

Figure 3: Realization of n input unitary gate using $n-1$ input unitary gates

3. Examination of the algorithm

The algorithm presented in the previous section has been tested for two types of states. In the first case we input to the register sampled Gaussian state of particle in the following form:

$$z_i = C \exp\left(-\frac{(x_i - \langle x \rangle)^2}{4dx^2} + \frac{i\langle p \rangle x_i}{\hbar}\right), \tag{6}$$

where $x_i = ix_{max}/n_b$ is i-th spatial sample, $\langle x \rangle$ is the expected value of the position, $\langle p \rangle$ is the expected value of the momentum, dx is standard deviation of the position while C is a normalization constant.
In the second case we input to the register sampled sine wave in the following form:

$$z_i = \begin{cases} \sqrt{2a^{-1}} \sin\left(\frac{\pi n x_i}{a}\right) & \text{for} \quad i = \frac{1}{2}n_b, \ldots \frac{3}{4}n_b \\ 0 & \text{for other,} \end{cases} \tag{7}$$

where $x_i = x_{max}(i/n_b - 0.5)$ and $n = 1, 2, \ldots$. Function given by Eq. (7) is sampled wave function of the paritcle which corresponds to the stationary state inside an infinite potential well of length $a = 0.25x_{max}$ located in $x \in (\frac{1}{2}x_{max}, \frac{3}{4}x_{max})$.
Results of the simulation are shown in Figs 4-6. In all cases $n_q = 8$. Fig. 4 shows result for the Gaussian state from Eq. (6) for $x_{max} = 20$nm, $dx = 0.02x_{max}$, $\langle x \rangle = 0.15x_{max}$. Parameter $\langle p \rangle$ corresponds to kinetic energy of the packet equal to 4eV. Fig. 5 shows result for the Gaussian state (Eq. (6)) for $x_{max} = 20$nm, $dx = 0.04x_{max}$, $\langle x \rangle = 0.8x_{max}$ and $\langle p \rangle$ corresponding to kinetic energy of the packet equal to 2eV. Fig. 6 shows results for state given by Eq. (7).

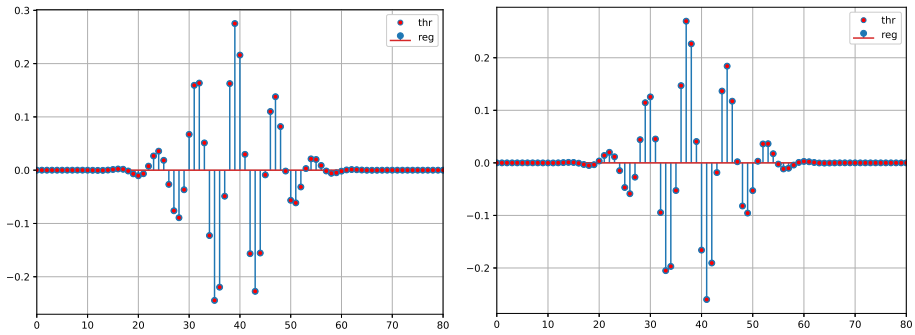

Figure 4: The state of the register after performing of the algorithm for the Gaussian function. The left plot shows real part of the state, while the right on shows an imaginary part. The figure shows only part of the state with non zero amplitudes. Red dots correspond to theoretical values given by Eq. (6), while blue vertical lines (ended with a blue dot) shows final state of the register.

In order to perform a quantitative analysis of the correctness of the obtained results, we introduce following errors:

$$\epsilon_1 = \sum_{i=0}^{n_b} |\psi_i - z_i|, \qquad \epsilon_2 = \sqrt{\sum_{i=0}^{n_b} |\psi_i - z_i|^2}, \tag{8}$$

where z_i are theoretical values (calculated from Eqs (6) and (7)) whereas ψ_i is the final state of the register. Values of errors (8) for the cases from Figs 4-6 are shown in Tab. 1.

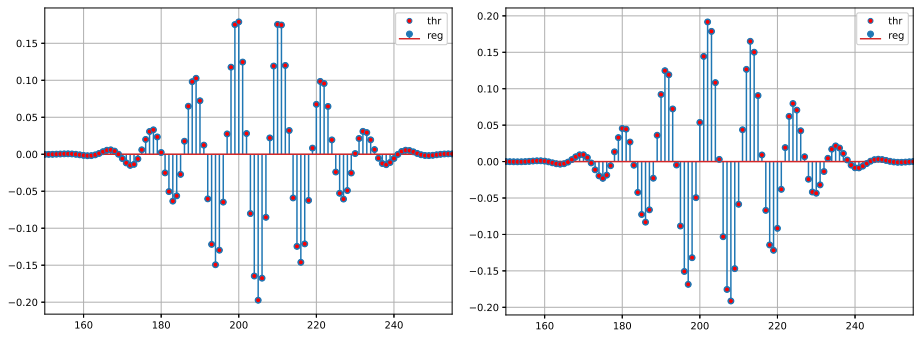

Figure 5: The state of the register after performing of the algorithm for the Gaussian function. The left plot shows real part of the state, while the right on shows an imaginary part. The figure shows only part of the state with non zero amplitudes. Red dots correspond to theoretical values given by Eq. (6), while blue vertical lines (ended with a blue dot) shows final state of the register.

4. Conclusions

- The main advantage of our algorithm is its universality. It allows one to enter into the register any arbitrary state. For this reason, the algorithm may be used not only in quantum simulation but also in quantum signal and image processing.
- It is simple algorithm based on dividing of probability. It can be implemented using standard quantum gates.
- The algorithm has been successfully tested using a Python library for quantum programming (PyQuil) - see Appendix.
- The main drawback of the presented algorithm is its computational complexity. We can conclude that the number of gates grows exponentially with the length of the register. Therefore, the algorithm is not suitable for large registers.

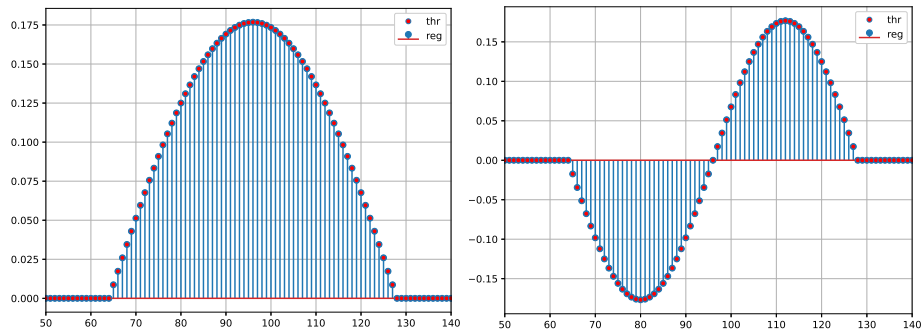

Figure 6: The state of the register after performing of the algorithm for sine function (Eq. (7)). The left plot shows real part of the state for $n = 1$, while the right on shows an real part for $n = 2$. The figure shows only part of the state with non zero amplitudes. Red dots correspond to theoretical values (Eq. (7)), while blue vertical lines (ended with a blue dot) shows final state of the register.

5. Appendix

Listing 1: Code of the algorithm in PyQuil

```
import math
import cmath
from pyquil import Program
from pyquil.gates import *

#symulation control:
nq=8#number of qubits
nb=2**nq#base states
#psi=...#nb-element list (final state of the register)

#probability list:
modz2 = list()
for i in range(nb): modz2.append(abs(psi[i])**2)

#phase list:
faza = list()
for i in range(nb): faza.append(cmath.phase(psi[i]))

#table of coefficients calculation (only numerators):
```

Table 1: Errors calculated for the cases from Figs 4-6

Fig. no.	ϵ_1	ϵ_2
4	$1.78 \cdot 10^{-8}$	$2.02 \cdot 10^{-7}$
5	$9.27 \cdot 10^{-15}$	$4.54 \cdot 10^{-14}$
6(left)	$3.13 \cdot 10^{-15}$	$3.78 \cdot 10^{-14}$
6(right)	$2.53 \cdot 10^{-15}$	$3.12 \cdot 10^{-14}$

```
Tab = list()
for i in range(0,nq,1):
    p=2**(i+1)
    q=nb//p
    podTab = list()
    for j in range(p):
        wsp=0
        for k in range(j*q,(j+1)*q,1): wsp+=modz2[k]
        podTab.append(wsp**0.5)
    Tab.append(podTab)

#table of coefficients (denominators):
for k in range(nq-1):
    for m in range(k+1, nq, 1):
        for j in range(2**(m+1)):
            if (Tab[k][j//(2**(m-k))]!=0): Tab[m][j]/=Tab[k][j//(2**(m-k))]

#table of angles (arccos):
TabPhi = list()
for i in range(nq):
    PodTabPhi = list()
    for j in range(0, 2**(i+1), 2):
        phi = math.acos(Tab[i][j])
        PodTabPhi.append(phi)
    TabPhi.append(PodTabPhi)

#program (amplitudes):
alg=Program()
for nu in range(nq, 0, -1):
    kb=2**(nq-nu)
    for i in range(nu+1, nq+1, 1): alg.inst(X(i))
    kontrolne = list(range(nu+1,nq+1,1))#list of control qubits
    for k in range(kb-1):
```

```
            alg.inst(RY(2*TabPhi[nq-nu][k], nu).controlled(kontrolne))
            alg.inst(X(nu+1))
            for dn in range(2, nq+1, 1):
                maska=2**(dn-1)
                if k%maska==(maska-1): alg.inst(X(nu+dn))
        alg.inst(RY(2*TabPhi[nq-nu][kb-1], nu).controlled(kontrolne))

#program (phases):
kb=2**(nq-1)
for i in range(2, nq+1, 1): alg.inst(X(i))
kontrolne = list(range(2,nq+1,1))#list of control qubits
for k in range(kb-1):
    alg.inst(X(1))
    alg.inst(PHASE(faza[2*k],1).controlled(kontrolne))
    alg.inst(X(1))
    alg.inst(PHASE(faza[2*k+1],1).controlled(kontrolne))
    alg.inst(X(2))
    for dn in range(2, nq+1, 1):
        maska=2**(dn-1)
        if k%maska==(maska-1): alg.inst(X(1+dn))
alg.inst(X(1))
alg.inst(PHASE(faza[2*kb-2],1).controlled(kontrolne))
alg.inst(X(1))
alg.inst(PHASE(faza[2*kb-1],1).controlled(kontrolne))
print(alg)
```

References

[1] Feynman, R., Internat. J. Theor. Phys., Vol. 21, 1982, pp. 467–488.

[2] Shor, P. W., Proc 35th Ann. Symp. Found. Comp. Sci., IEEE Comp.Soc. Pr., Vol. 124, 1994.

[3] Grover, L. K., *From Schrodinger equation to the quantum search algorithm*, Am. J. Phys., Vol. 69, 2001, pp. 769–777.

[4] Lloyd, S., *Universal Quantum Simulators*, Science, Vol. 273, 1996, pp. 5278.

[5] Schaetz, T., Monroe, C. R., and Esslinger, T., *Focus on quantum simulation*, New Journal of Physics, Vol. 15, 2013, pp. 085009.

[6] Lanyon, B. P., *Universal digital quantum simulation with tapped ions*, 2011, http://xxx.lanl.gov//arXiv:1109.1512v2.

[7] Childs, A. M., Maslov, D., Nam, Y., Ross, N. J., and Su, Y., *Toward the first quantum simulation with quantum speedup*, PNAS, Vol. 115, No. 38, 2018.

[8] Wecker, D., *Solving strongly correlated electron models on a quantum computer*, Phys Rev A, Vol. 92, 2015, pp. 062318.

[9] Kokail, C., Maier, C., and van Bijnen, R., *Self-verifying variational quantum simulation of lattice models*, Nature, Vol. 569, 2019.

[10] Wecker, D., Bauer, B., Clark, B. K., Hastings, M. B., and Troyer, M., *Gate count estimates for performing quantum chemistry on small quantum computers*, Phys Rev A, Vol. 90, 2014, pp. 022305.

[11] Hempel, C., Maier, C., and Romero, J., *Quantum Chemistry Calculations on a Trapped-Ion Quantum Simulator*, Phys. Rev. X, Vol. 8, 2018, pp. 031022.

[12] Jordan, S. P., Lee, K. S. M., and Preskill, J., *Quantum algorithms for quantum field theories*, Science, Vol. 336, 2012, pp. 1130–1133.

[13] Ostrowski, M., *Simulation of the Schrödinger particle nonelastic scattering with emission of photon in the quantum register*, Bull. Pol. Ac.: Tech., Vol. 68, No. 5, 2020.

[14] Ostrowski, M., *Simulation of diffusion of a single Schrödinger particle in the quantum register*, Acta Phys. Polon. A, Vol. 137, No. 6, 2020, pp. 1182–1186.

[15] Ventura, D., *Learning quantum operators*, In: Proceedings of the Joint Conference on Information Sciences, 2000, pp. 750–752.

[16] Nielsen, M. A. and Chuang, I. L., *Quantum Computation and Quantum Information*, Cambridge University Press, 2000.

[17] Faber, J., Thess, R. N., and Giraldi, G., *Lerning linear operators by generic algorithms*, 2003.

[18] Rubinstein, B. I. P., *Evolving quantum circuits using generic programming*, In: Generic Algorithms and Generic Programming at Stanford 2000, Stanford Bookstore, Stanford, California, 94305-3079 USA, 2000, pp. 325–334.

[19] Williams, C. P. and Gray, A., *Automated Design of Quantum Circuits*, Lecture Notes in Computer Science, Springer-Verlag New York, Inc., Vol. 1509, 1999.

[20] Siedlecka-Lamach, O., *A minimization algorithm of 1-way quantum finite automata*, Metody Informatyki Stosowanej, Polska Akademia Nauk Oddzial w Gdansku, Komisja Informatyki, , No. 4, 2010, pp. 73–79.

[21] Agaian, S. S. and Klappenecker, A., *Quantum Computing and a Unitary Approach to Fast Unitary Transforms*, Image Processing: Algorithms and Systems, 2002.

[22] Hoyer, P., *Efficient Quantum Transforms*, http://arXiv:quant-ph/9702028, 1997.

[23] Pang, C.-Y. and Zhou, R.-G., *Signal and image compression using quantum discrete cosine transform*, Information Sciences, Vol. 473, 2019, pp. 121–141.

[24] Preskill, J., http://www.theory.caltech.edu/~preskill/ph229.