JOURNAL

OF APPLIED COMPUTER SCIENCE & MATHEMATICS

No 19 (9) / 2015

Journal of Applied Computer Science & Mathematics is published biannually by the "Ştefan cel Mare" University of Suceava, Romania.
The goal of journal covers: mathematics, applied computer science, computer networks and telecommunications. The authors are responsible for the content of their contributions.

Contact
"Ştefan cel Mare" University of Suceava
Universității nr.13, Corp H
720229 Suceava, Romania

http://www.jacs.usv.ro
e-mail: dtiliute@seap.usv.ro
paulp@seap.usv.ro

Journal of Applied Computer Science & Mathematics
ISSN: 2066-4273

JOURNAL
OF APPLIED COMPUTER SCIENCE & MATHEMATICS

No 19 (9) / 2015

Editura Universității Suceava

Cover design and layout: Paul Paşcu

Suceava, 2015

TABLE OF CONTENTS

Foreword 7

What the Current System Development Trends tell us about Systems Development Methodologies: Toward explaining SSDAM, Agile and IDEF0 Methodologies 9
Abdulla F. ALLY, Zhang NING

Strength Pareto Evolutionary Algorithm using Self-Organizing Data Analysis Techniques 16
Ionuț BĂLAN

Quantum Computing - A new Implementation of Simon Algorithm for 3-Dimensional Registers 23
Adina BĂRÎLĂ

An Evaluation of CAPM's validity in the Romanian Stock Exchange 31
Camelia COLESCU, Elena-Ariadna PAPUC

Improving the Security of Internet Banking Applications by Using Multimodal Biometrics 37
Cătălin LUPU, Vasile-Gheorghiță GĂITAN, Valeriu LUPU

Evolutionary Approach Based on Active Edges Detection for Images Segmentation 43
Slatnia SIHEM, Kazar OKBA, Arab KHAOULA

A Proposed Method for Pattern Classification with HMM in the Context of Supervised Learning 50
Ştefan Gheorghe PENTIUC, Ştefania - Iuliana ŞOIMAN

Author's Index 57

Foreword

A New Tool for Researchers
ResearchGate – The Largest Free Professional Network for Researchers

Creating a social network is not a revolutionary idea anymore. There are social networks on every subject nowadays: music, film, cars or only for socializing with friends. What about a social network targeting a specific group of people like researchers and scientists?

Researchers often meet with obstacles delaying their research progress, even if there has eventually already been found a solution somewhere else in the world. In order to facilitate the progress in science by sharing knowledge all over the world, was launched in 2008 ResearchGate, the largest academic scientific network, by three young researchers.

The Online Platform gathers more the 800.000 researchers and scientists all over the world. It's founders, researchers themselves, knew researchers need and designed the site according to this. Users have the opportunity not only to find fellow researchers and collaborate, but also to find relevant literature, to create discussion groups on their research interests, to find jobs in their field or to publish their papers. "Our scientific community connects scientists and researchers from Africa, South America or Russia, countries between which there has never been a scientific collaboration before.", says Ijad Madisch, founder and CEO of ResearchGate.

ResearchGate is specially designed for the needs of scientists: starting from the profile, containing information about researchers projects, publications etc. to the semantic search algorithm in the similar abstract search, finding related articles within a database of more than 30 million documents. For example, by tipping in "computer science" you get more than 50 researchers in this field, more than 25 discussion groups, as well as more than 10.000 publications. Hence ResearchGate provides a collaborative working space without any spatially or temporarily borders.

To discover and benefit from ResearchGate facilities go to www.researchgate.net

Editorial Board

Author's Index

ALLY F. Abdulla, 9

BĂLAN Ionuț, 16

BĂRÎLĂ Adina, 23

COLESCU Camelia, 31

GĂITAN Vasile - Gheorghiță, 37

KHAOULA Arab, 43

LUPU Cătălin, 37

LUPU Valeriu, 37

NING Zhang, 9

OKBA Kazar, 43

PAPUC Elena-Ariadna, 31

PENTIUC Ştefan - Gheorghe, 50

SIHEM Slatnia, 43

ŞOIMAN Ştefania - Iuliana, 50

Quantum Computing - A new Implementation of Simon Algorithm for 3-Dimensional Registers

Adina BĂRÎLĂ
“Ştefan cel Mare” University of Suceava, Romania
adina@eed.usv.ro

***Abstract*–Quantum computing is a new field of science aiming to use quantum phenomena in order to perform operations on data. The Simon algorithm is one of the quantum algorithms which solves a certain problem exponentially faster than any classical algorithm solving the same problem. Simulating of quantum algorithms is very important since quantum hardware is not available outside of the research labs. QCL (Quantum Computation Language) is the most advanced implemented quantum computer simulator and was conceived by Bernhard Ömer. The paper presents an implementation in QCL of the Simon algorithm in the case of 3-dimensional registers**.

Keywords: **quantum computing, quantum gate, quantum algorithm.**

I. Introduction

Quantum computing is a new field of science whose origin is the Richard Feynman’s idea for constructing a computer to simulate the quantum systems [1]. Introduced in the early 1980’s, quantum computing investigates the computational power of computer based on quantum mechanical principles and wants to find algorithms faster than classical algorithms solving the same problem. David Deutsch introduced two models for quantum computation: a quantum version of Turing machine [2] and quantum circuits [3]. He demonstrated that the universal quantum computer can do things that the universal Turing machine cannot. He also demonstrated that quantum gates can be combined to achieve quantum computation in the same way that Boolean gates can be combined to achieve classical computation.

David Deutsch invented the first quantum algorithm which solves a computational problem in a more efficient way that classical computation. He presented an example which showed that a single quantum computation may suffice to decide whether a given one-bit function is constant or balanced. The Deutsch-Jozsa algorithm was designed in 1992 to maximally illustrate the computational advantage of quantum computing over classical computing. Other notable algorithms were developed by Simon and Vazirani. But the most important results in the field of quantum computing are considered the Shor’s and the Grover’s algorithms. In 1994, Peter Shor described a polynomial time quantum algorithm for factoring integers[4] and in 1996 Lov Grover invented the quantum database search algorithm which achieved quadratic speedup for the classic problem of database search [5]. From those years, the research in quantum computing field has accelerated, computer scientists trying to build quantum computers and find other quantum algorithms.

This paper aims to present an original implementation of Simon algorithm based on Quantum Computation Language (QCL). Section II presents basic concepts of quantum computation. Section III introduces Simon algorithm and section IV presents a QCL implementation of this . Section V draws some conclusion and future work.

II. Quantum Computation – Basic Concepts

The fundamanetal unit of quantum information is called quantum bit or qubit [6]. A qubit is a physical system which has two basis states, conventionally written $|0\rangle$ and $|1\rangle$, corresponding to the classical values 0 and 1. Unlike the classical bit, the general state of a qubit is a linear combination – or a superposition – of the basis states:

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle \tag{1}$$

where the amplitudes α and β are complex numbers such that:

$$|\alpha|^2+|\beta|^2=1 \tag{2}$$

In other words, a qubit can exist as a zero, a one, or simultaneously as both 0 and 1 (when both α and β are nonzero). Formally, a quantum state is a unit vector in a Hilbert space.

A system consisting of n qubits has 2^n basis states and its general state is a superposition of all basis states:

$$|\psi\rangle = \sum_{k=0}^{2^n-1} c_k |k\rangle \tag{3}$$

where:

$$|k\rangle = |k_{n-1}\ldots k_1 k_0\rangle \tag{4}$$

with $|k_j\rangle$ represents the state of qubit j and $|k_{n-1}\ldots k_1 k_0\rangle$ (or $|k_{n-1}\rangle\ldots|k_1\rangle|k_0\rangle$ or $|k_{n-1},\ldots,k_1,k_0\rangle$) represents the tensor product $\left|k_{n-1}\right\rangle \otimes \ldots \otimes \left|k_1\right\rangle \otimes \left|k_0\right\rangle$. The amplitudes c_k are complex numbers such that:

$$\sum_{k=0}^{2^n-1} |c_k|^2 = 1 \tag{5}$$

Like the single-qubit system, a n-qubit register can store simultaneously all the basis states. A state of a n-qubit register is an element in the space $H^n = H\otimes H\otimes\ldots\otimes H$ (tensor product).

Evolution of a quantum system can be described by a unitary transformation U. A unitary transformation that acts on a small number of qubits is called a gate. A quantum gate has the same number of inputs and outputs. A one-qubit elementary gate is described by a 2x2 matrix:

$$U = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \tag{6}$$

which transforms |0⟩ into a|0⟩+b|1⟩ and |1⟩ into c|0⟩+d|1⟩. The Hadamard (H) and the Pauli (X,Y,Z) gates are examples of quantum gates that act on a single qubit:

$$H = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \qquad X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \tag{7}$$

$$Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} \qquad Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \tag{8}$$

The most important two-qubit gate is the CNOT (controlled-not gate). It has two input qubits, the control and the target qubit. The target qubit is flipped only if the control qubit is set to 1. The matrix form of this gate is:

$$CNOT = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \tag{9}$$

and the circuit representation is represented in fig.1.

CNOT is a generalization of the classical XOR gate, since its action may be summarized as $|x,y\rangle \rightarrow |x, y\oplus x\rangle$, where $\oplus$ is addition modulo two, which is the same as XOR.

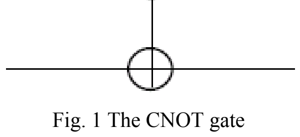


Fig. 1 The CNOT gate

Generally, if U is a one-qubit gate with matrix representation:

$$U = \begin{pmatrix} x_{00} & x_{01} \\ x_{10} & x_{11} \end{pmatrix} \tag{10}$$

then the controlled-U gate is a two-qubit gate with matrix representation:

$$C(U) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & x_{00} & x_{01} \\ 0 & 0 & x_{10} & x_{11} \end{pmatrix} \tag{11}$$

The first qubit is the control qubit.

The SWAP gate is the quantum generalisation of the CROSSOVER classical gate. It swaps the quantum states of the qubits. The matrix representation is:

$$SWAP = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{12}$$

The Cph (controlled phase) gate acts on two qubits and it has no classical equivalent.

$$U_{cph}(\phi) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & \exp(i\phi) \end{pmatrix} \tag{13}$$

An important three-qubit gate is the CCNOT (controlled-controlled-not) gate. It has two control qubits and a target qubit. The target qubit is flipped only if the control qubits are set to 1. The matrix form of CCNOT gate is given in eqn. 14 and the circuit representation is shown in fig. 2.

$$CNOT = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} \tag{14}$$

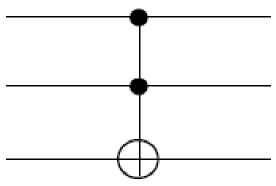


Fig. 2 The CCNOT gate

Measurement is the only nonreversible operation which can be applied to a quantum state. Measurement collapses a quantum state into one of the possible basis states, so measurement is a destructive operation. If a qubit is in the state |ψ⟩ = α|0⟩ + β|1⟩ and a measure is performed, it obtaines 0 with probability α^2 (the qubit's state becomes |0⟩) and 1 with probability β^2 (the qubit's state becomes |1⟩).

III. SIMON ALGORITHM

A. Overview

Daniel Simon [7] proposed the following problem: let f be a function of the form:

f: {0,1}n → {0,1}n

for a positive integer n. The function f is promised to have the property that there exists a string s ϵ {0,1}n, s≠0 such that:

∀x,y ϵ {0,1}n, f(x) = f(y) ⇔ y=x⊕ s

The goal of the problem is to find the period s.

For example, if n=3, the following function satisfies the required property:

x	*f(x)*
000	100
001	010
010	000
011	110
100	000
101	110
110	100
111	010

Specifically, the string s is 110.

A quantum algorithm for solving this problem has a quantum part and a classical post-processing part [8]. The quantum part consists of following steps:

In the first step, two n-qubit registers are initialized to |0⟩n = |00…00□.

So, the initial state of the system is:

$$|\psi_0\rangle = |0\rangle_n |0\rangle_n \tag{15}$$

The second step consist of applying the $H^{\otimes n}$ transform to the first register. $H^{\otimes n}$ stands for *H*⊗*H*⊗…⊗*H* (where *H* is the Hadamard transform). The *H* transform can be generalized on *n* qubits like in the following [9]:

$$H^{\otimes n}|x\rangle_n = \frac{1}{\sqrt{2^n}}\sum_{y=0}^{2^n-1}(-1)^{x\cdot y}|y\rangle_n \tag{16}$$

where the product *x·y* is defined as:

$$x\cdot y = x_1\cdot y_1 \oplus x_2\cdot y_2 \oplus \ldots \oplus x_n\cdot y_n$$

After the $H^{\otimes n}$ transform is performed, the quantum state becomes:

$$|\psi_1\rangle = \left(H^{\otimes n}\otimes I_{2^n}\right)|0\rangle_n|0\rangle_n = \frac{1}{2^{n/2}}\sum_{x=0}^{2^n-1}|x\rangle_n|0\rangle_n \tag{17}$$

In the next step, the oracle transform U_f acts on both registers. The U_f transformation is defined by:

$$U_f|x\rangle_n |y\rangle_n = |x\rangle_n |f(x)\oplus y\rangle_n \tag{18}$$

where ⊕ denotes the bitwise XOR.

The quantum state becomes:

$$|\psi_2\rangle = U_f\left(\frac{1}{2^{n/2}}\sum_{x=0}^{2^n-1}|x\rangle_n|0\rangle_n\right) = \frac{1}{2^{n/2}}\sum_{x=0}^{2^n-1}|x\rangle_n|f(x)\rangle_n \tag{19}$$

Finally, $H^{\otimes n}$ transform is applied on first register and the quantum state becomes:

$$|\psi_3\rangle = \left(H^{\otimes n}\otimes I_{2^n}\right)|\psi_2\rangle = \frac{1}{2^{n/2}}\sum_{x=0}^{2^n-1}H^{\otimes n}|x\rangle_n|f(x)\rangle_n \tag{20}$$

According to eq (16), the final state can be written as:

$$|\psi_3\rangle = \frac{1}{2^n}\sum_{x,y=0}^{2^n-1}(-1)^{x\cdot y}|y\rangle_n|f(x)\rangle_n \tag{21}$$

$$|\psi_3\rangle = \sum_{y=0}^{2^n-1}|y\rangle_n\left(\frac{1}{2^n}\sum_{x=0}^{2^n-1}(-1)^{x\cdot y}|f(x)\rangle_n\right)$$

Let A = range(f) and let z϶A. By the definition of the function f, there are exactly two possible values xz, x'z϶ {0,1}n such that

f(xz) = f(x'z) = z,

and moreover x'z = xz⊕s. So,

$$\begin{aligned}|\psi_3\rangle &= \sum_{y=0}^{2^n-1}|y\rangle_n\left(\frac{1}{2^n}\sum_{z\in A}\left((-1)^{x_z\cdot y}+(-1)^{x'_z\cdot y}\right)|z\rangle_n\right)\\ |\psi_3\rangle &= \sum_{y=0}^{2^n-1}|y\rangle_n\left(\frac{1}{2^n}\sum_{z\in A}\left((-1)^{x_z\cdot y}+(-1)^{(x_z\oplus s)\cdot y}\right)|z\rangle_n\right)\\ |\psi_3\rangle &= \sum_{y=0}^{2^n-1}|y\rangle_n\left(\frac{1}{2^n}\sum_{z\in A}(-1)^{x_z\cdot y}\left(1+(-1)^{(s\cdot y)}\right)|z\rangle_n\right)\end{aligned} \tag{22}$$

Now the value of the first register is measured.
In the case where s ≠ 0^n, probability to measure a value *y* is:

$$\begin{aligned}P(y) &= \sum_{z\in A}\left|\frac{1}{2^n}(-1)^{x_z\cdot y}\left(1+(-1)^{(s\cdot y)}\right)\right|^2\\ &= \begin{cases}\frac{1}{2^{n-1}} & if\ s\cdot y=0\\ 0 & if\ s\cdot y=1\end{cases}\end{aligned} \tag{23}$$

So, the measurement always results in a value that satisfies s·y = 0.

Measurement of the first register will give a y1϶{0,1}n where y1·s=0. The algorithm is restarted and a new measurement will give a new value, y2϶{0,1}n where y2·s=0, y2≠y1 and y2≠0. s is uniquely determined once we have n−1 linearly independent equations [10]. Simon's algorithm is repeated n-1 times to obtain a system of n-1 linear equations of the form:

$$\begin{gathered}y_1\cdot s = 0\\ y_2\cdot s = 0\\ \dots\\ y_{n-1}\cdot s = 0\end{gathered}$$

The classical post-processing part consists of solving this system of equations in *n* unknowns (the bits of *s*) to find *s*.

B.The Simon's algorithm for n=3

In the case where n = 3, the quantum circuit has two registers of size 3 and both are initialized to the state |000⟩.

The |ψ₀⟩, |ψ₁⟩, |ψ₂⟩, |ψ₃⟩ quantum states are presented in the Appendix A. The final forms of these states are:

$$|\psi_0\rangle = |000\rangle|000\rangle$$

$$\begin{aligned}|\psi_1\rangle = \frac{1}{2\sqrt{2}}\big(&|000\rangle|000\rangle+|001\rangle|000\rangle+|010\rangle|000\rangle+\\ &+|011\rangle|000\rangle+|100\rangle|000\rangle+|101\rangle|000\rangle+\\ &+|110\rangle|000\rangle+|111\rangle|000\rangle\big)\end{aligned}$$

$$\begin{aligned}|\psi_2\rangle = \frac{1}{2\sqrt{2}}\big(&|000\rangle|100\rangle+|001\rangle|010\rangle+|010\rangle|000\rangle+\\ &+|011\rangle|110\rangle+|100\rangle|000\rangle+|101\rangle|110\rangle+\\ &+|110\rangle|100\rangle+|111\rangle|010\rangle\big)\end{aligned}$$

$$\begin{aligned}|\psi_3\rangle = \frac{1}{2^2}\big(&|000\rangle|000\rangle+|000\rangle|010\rangle+|000\rangle|100\rangle+\\ &+|000\rangle|110\rangle+|001\rangle|000\rangle-|001\rangle|010\rangle+\\ &+|001\rangle|100\rangle-|001\rangle|110\rangle-|110\rangle|000\rangle\\ &+|110\rangle|010\rangle+|110\rangle|100\rangle-|110\rangle|110\rangle\\ &-|111\rangle|000\rangle-|111\rangle|010\rangle+|111\rangle|100\rangle-\\ &+|111\rangle|110\rangle\big)\end{aligned}$$

In order to implement the algorithm in QCL, the U_f transformation must be described by quantum gates. According to the definition of U_f transformation (relation (18)), in the example given above, the action of U_f in the case where the state of the first register is |000⟩ can be described as:

$$\begin{aligned}U_f|000\rangle|000\rangle &= |000\rangle|000\oplus f(000)\rangle = |000\rangle|000\oplus 100\rangle = |000\rangle|100\rangle\\ U_f|000\rangle|001\rangle &= |000\rangle|001\oplus f(000)\rangle = |000\rangle|001\oplus 100\rangle = |000\rangle|101\rangle\\ U_f|000\rangle|010\rangle &= |000\rangle|010\oplus f(000)\rangle = |000\rangle|010\oplus 100\rangle = |000\rangle|110\rangle\\ U_f|000\rangle|011\rangle &= |000\rangle|011\oplus f(000)\rangle = |000\rangle|011\oplus 100\rangle = |000\rangle|111\rangle\\ U_f|000\rangle|100\rangle &= |000\rangle|100\oplus f(000)\rangle = |000\rangle|100\oplus 100\rangle = |000\rangle|000\rangle\\ U_f|000\rangle|101\rangle &= |000\rangle|101\oplus f(000)\rangle = |000\rangle|101\oplus 100\rangle = |000\rangle|001\rangle\\ U_f|000\rangle|110\rangle &= |000\rangle|110\oplus f(000)\rangle = |000\rangle|110\oplus 100\rangle = |000\rangle|010\rangle\\ U_f|000\rangle|111\rangle &= |000\rangle|111\oplus f(000)\rangle = |000\rangle|111\oplus 100\rangle = |000\rangle|011\rangle\end{aligned}$$

If the state of the first register is |001⟩, U_f acts as follows:

$$\begin{aligned}U_f|001\rangle|000\rangle &= |001\rangle|000\oplus f(001)\rangle = |001\rangle|000\oplus 010\rangle = |001\rangle|010\rangle\\ U_f|001\rangle|001\rangle &= |001\rangle|001\oplus f(001)\rangle = |001\rangle|001\oplus 010\rangle = |001\rangle|011\rangle\\ U_f|001\rangle|010\rangle &= |001\rangle|010\oplus f(001)\rangle = |001\rangle|010\oplus 010\rangle = |001\rangle|000\rangle\\ U_f|001\rangle|011\rangle &= |001\rangle|011\oplus f(001)\rangle = |001\rangle|011\oplus 010\rangle = |001\rangle|001\rangle\\ U_f|001\rangle|100\rangle &= |001\rangle|100\oplus f(001)\rangle = |001\rangle|100\oplus 010\rangle = |001\rangle|110\rangle\\ U_f|001\rangle|101\rangle &= |001\rangle|101\oplus f(001)\rangle = |001\rangle|101\oplus 010\rangle = |001\rangle|111\rangle\\ U_f|001\rangle|110\rangle &= |001\rangle|110\oplus f(001)\rangle = |001\rangle|110\oplus 010\rangle = |001\rangle|100\rangle\\ U_f|001\rangle|111\rangle &= |001\rangle|111\oplus f(001)\rangle = |001\rangle|111\oplus 010\rangle = |001\rangle|101\rangle\end{aligned}$$

If the state of the first register is $|101\rangle$, U_f acts as follows:

$$U_f|101\rangle|000\rangle=|101\rangle|000\oplus f(101)\rangle=|101\rangle|000\oplus 110\rangle=|101\rangle|110\rangle$$
$$U_f|101\rangle|001\rangle=|101\rangle|001\oplus f(101)\rangle=|101\rangle|001\oplus 110\rangle=|101\rangle|111\rangle$$
$$U_f|101\rangle|010\rangle=|101\rangle|010\oplus f(101)\rangle=|101\rangle|010\oplus 110\rangle=|101\rangle|100\rangle$$
$$U_f|101\rangle|011\rangle=|101\rangle|011\oplus f(101)\rangle=|101\rangle|011\oplus 110\rangle=|101\rangle|101\rangle$$
$$U_f|101\rangle|100\rangle=|101\rangle|100\oplus f(101)\rangle=|101\rangle|100\oplus 110\rangle=|101\rangle|010\rangle$$
$$U_f|101\rangle|101\rangle=|101\rangle|101\oplus f(101)\rangle=|101\rangle|101\oplus 110\rangle=|101\rangle|011\rangle$$
$$U_f|101\rangle|110\rangle=|101\rangle|110\oplus f(101)\rangle=|101\rangle|110\oplus 110\rangle=|101\rangle|000\rangle$$
$$U_f|101\rangle|111\rangle=|101\rangle|111\oplus f(101)\rangle=|101\rangle|111\oplus 110\rangle=|101\rangle|001\rangle$$

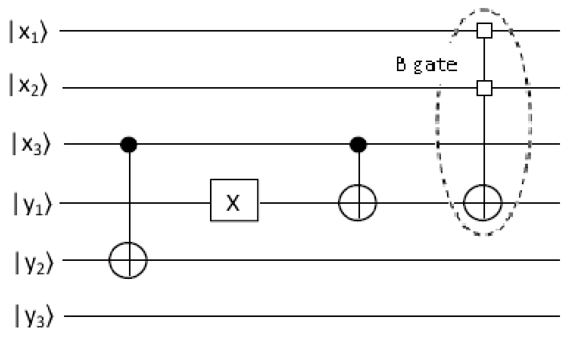


Fig.3.U_f transformation for Simon's algorithm

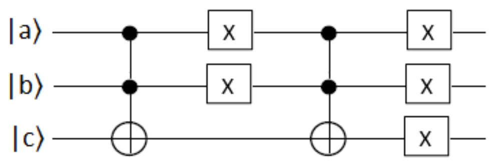


Fig.4 The B gate

Similarly, the action of U_f can be described for the other states of the registers.

The action of U_f can be represented as a 64 × 64 matrix and can be reproduced by a set of quantum gates (CNOT, X, CNOT and B) as shown in fig. 3.

In order to simulate the action of U_f, the author defined a gate (B gate) which acts on three qubits a, b, c and flips the third qubit (c) if only one of the first two qubits is set to 1 (if *a XOR b*). This gate can be written using CCNOT and X gates as drawn in fig. 4.

IV. The QCL implementation

QCL (Quantum Computation Language) is a high-level, architecture independent programming language for quantum computers [11]. It was conceived by Bernhard Ömer and the current version appeared in 2004. QCL was implemented in C, as a standalone full integrated compiler and runs under Linux operating system. Its syntax and data types are similar to those in C. The basic built-in quantum data type is `qreg` (quantum register), which can be interpreted as an array of qubits.

The main features of QCL are [12], [13]:

a) a classical control language with functions, flow-control, interactive I/O and various classical data types (`int, real, complex, boolean, string`);

b) 2 quantum operator types: general unitarian (`operator`) and reversible pseudo-classic gates (`qufunct`);

c) inverse execution, allowing for on-the-fly determination of the inverse operator though caching of operator calls;

d) various quantum data types (qubit registers) for compile time information on access modes (`qureg, quconst, quvoid, quscratch`);

e) convenient functions to manipulate quantum registers (`q[n]` – qubit, `q[n:m]` – substring, `q&p` – combined register);

f) quantum memory management (`quheap`) allowing for local quantum variables

g) easy adaptation to individual sets of elementary operators.

The QCL implementation of B quantum gate is presented below:

```
//B gate
operator B(qureg x, qureg y, qureg z)
{
    CCNot(x,y,z);
    Not(x);
    Not(y);
    CCNot(x,y,z);
    Not(x);
    Not(y);
    Not(z);
}
```

The QCL implementation of Simon's algorithm (the quantum part) is presented below:

```
operator simonGates(qureg x, qureg y)
{
  //the Hadamard transformation is applied
  H(x);
  //the Uf transformation is performed
  CNot(y[1],x[0]);
  Not(y[2]);
  CNot(y[2],x[0]);
  B(x[2],x[1],y[2]);
  //the Hadamard transformation is applied
  H(x);
}

procedure simon()
{
  qureg x[3]; //the first register
  qureg y[3]; //the second register
  int m1; int m2; //the measured values

  {
  reset;
  simonGates(x,y);
  //measure the x register
  measure x,m1;
  print "The first measured value is: ", m1;
  } until (m1!=0);

  //the algorithm is restarted
  //to obtain the second equation
  //the new measured value must be
  //different from the first measured value
  {
  reset;
  simonGates(x,y);
  //measure the x register
  measure x,m2;
  print "The second measured value is: ", m2;
  } until ((m2!=0) and (m2!=m1));
}
```

Solving the system of equations is the classical post-processing part of the algoritm. The implemetantion of this part, also in QCL, is presented in the Appendix B.

In the figure 5 it can be seen various values measured at various program executions. At the first run of the program the measured values are 6 (110_2) and 1 (001_2). So, the system in 3 unknowns to be solved is:

$$s_1 \cdot 1 + s_2 \cdot 1 + s_3 \cdot 0 = 0$$
$$s_1 \cdot 0 + s_2 \cdot 0 + s_3 \cdot 1 = 0$$

where s_1, s_2, s_3 are the bits of string s and all of the operations are modulo 2 operations. This system has two solutions:

$$s_1 = s_2 = s_3 = 0$$

and

$$s_1 = s_2 = 1, s_3 = 0$$

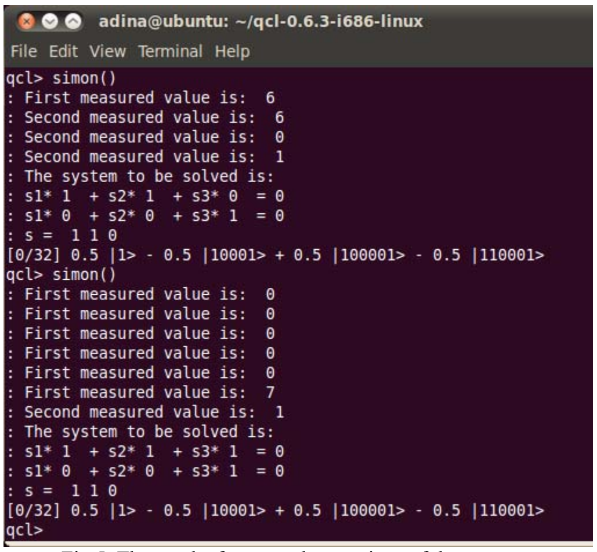
But it was supposed $s \neq 0$, so the only valid solution is $s = 110$ (6_{10}).

On the last line QCL displays the current state of the quantum machine.

V. Conclusions and future work

Quantum computing permits to perform computational operations on data much faster and efficiently by taking advantage of quantum parallelism. At the same time, by using the principle of superposition, a large amount of data could be stored. In absence of quantum devices, quantum computing simulators helps programmers to exploit the features of quantum computers and understand the constraints imposed by these devices. In the last years many quantum computing simulators have been developed in order to simulate quantum algorithms. In this paper the QCL quantum language has been used to simulate the quantum algorithm developed by David Simon and known as Simon's algorithm. This has been implemented for the case of 3-dimensional registers.

This paper is a first attempt to develop a QCL implementation of Simon algorithm. The oracle transformation was simulated by CNOT gates, X gates and a new 3-qubit gate which flips the third qubit if only one of the first two qubits is set to 1. Further we will implement this algorithm for other dimensions of the input registers and other functions.



adina@ubuntu: ~/qcl-0.6.3-i686-linux
File Edit View Terminal Help

```
qcl> simon()
: First measured value is:  6
: Second measured value is:  6
: Second measured value is:  0
: Second measured value is:  1
: The system to be solved is:
: s1* 1  + s2* 1  + s3* 0  = 0
: s1* 0  + s2* 0  + s3* 1  = 0
: s =  1 1 0
[0/32] 0.5 |1> - 0.5 |10001> + 0.5 |100001> - 0.5 |110001>
qcl> simon()
: First measured value is:  0
: First measured value is:  0
: First measured value is:  0
: First measured value is:  0
: First measured value is:  0
: First measured value is:  7
: Second measured value is:  1
: The system to be solved is:
: s1* 1  + s2* 1  + s3* 1  = 0
: s1* 0  + s2* 0  + s3* 1  = 0
: s =  1 1 0
[0/32] 0.5 |1> - 0.5 |10001> + 0.5 |100001> - 0.5 |110001>
qcl>
```

Fig.5. The results for several executions of the program

Acknowledgment

This paper was supported by the project "Sustainable performance in doctoral and post-doctoral research PERFORM - Contract no. POSDRU/159/1.5/S/138963", project co-funded from European Social Fund through Sectorial Operational Program Human Resources 2007-2013.

References

[1] R. Feynman, "Simulating physics with computers", International Journal of Theoretical Physics, vol. 21, no. 6, pages 467–488, 1982

[2] D. Deutsch, "Quantum theory, the Church-Turing principle and the universal quantum computer", Proceedings of the Royal Society of London A 400, pp. 97-117, 1985

[3] D. Deutsch, "Quantum computational networks", Proceedings of the Royal Society of London A 425, pp. 73-90, 1989

[4] P.W. Shor, "Algorithms for Quantum Computing: Discrete Logarithm and Factoring", Proceedings of 35th Annual Symposium on Foundations of Computer Science, Los Alamitos, CA, USA, 1994, pp. 124-134

[5] L.K.Grover, "A fast quantum mechanical algorithm for database search", Proc. 28th Annual ACM Symposium on the Theory of Computing (STOC), 1996, p. 212-219

[6] B. Schumacher, "Quantum coding", Physical Review A, Vol. 51, No. 4, April 1995

[7] D. R. Simon, "On the Power of Quantum Computation",SIAM Journal on Computing, no. 5, p. 1474.
[8] John Watrous, *Lecture Notes on Quantum Computing*, University of Waterloo, 2006
[9] D. Mermin, *Lectures Notes on Quantum Computer*. Cornell University, Ithaca, New York, 2006.
[10] Umesh Vazirani, *Lecture Notes on Quantum Computing*, University of California, Berkely, 2007
[11] H. De Raedt, K. Michielsen, *Computational Methods for Simulating Quantum Computers,* arXiv:quant-ph/0406210, 2004
[12] B. Ömer, *Quantum Programming in QCL,* Technical University of Vienna, Austria, 2000.
[13] B. Ömer, *Strucured Quantum Programming in QCL,* Technical University of Vienna, Austria, 2003.

Adina Bărîlă is a PhD student at „Ștefan cel Mare" University of Suceava in Computers and Information Technology area. Her research interests include quantum computing and databases.

Appendix A

In the case where n = 3 and U_f is the oracle transform corresponding to function given in this paper , the quantum states $|\psi_0\rangle$, $|\psi_1\rangle$, $|\psi_2\rangle$, $|\psi_3\rangle$ are:

$$|\psi_0\rangle = |000\rangle|000\rangle$$

$$|\psi_1\rangle = \left(H^{\otimes 3} \otimes I_{2^3}\right)|000\rangle|000\rangle = \left(H^{\otimes 3}|000\rangle\right)|000\rangle$$

$$H^{\otimes 3}|000\rangle = H|0\rangle H|0\rangle H|0\rangle = \frac{1}{\sqrt{2}}\left(|0\rangle+|1\rangle\right)\frac{1}{\sqrt{2}}\left(|0\rangle+|1\rangle\right)\frac{1}{\sqrt{2}}\left(|0\rangle+|1\rangle\right) = \frac{1}{2\sqrt{2}}\left(|000\rangle+|001\rangle+|010\rangle+|011\rangle+|100\rangle+|101\rangle+|110\rangle+|111\rangle\right)$$

$$|\psi_1\rangle = \frac{1}{2\sqrt{2}}\left(|000\rangle|000\rangle+|001\rangle|000\rangle+|010\rangle|000\rangle+|011\rangle|000\rangle+|100\rangle|000\rangle+|101\rangle|000\rangle+|110\rangle|000\rangle+|111\rangle|000\rangle\right)$$

$$|\psi_2\rangle = U_f|\psi_1\rangle = \frac{1}{2\sqrt{2}}\left(|000\rangle|100\rangle+|001\rangle|010\rangle+|010\rangle|000\rangle+|011\rangle|110\rangle+|100\rangle|000\rangle+|101\rangle|110\rangle+|110\rangle|100\rangle+|111\rangle|010\rangle\right)$$

$$\begin{aligned}|\psi_3\rangle = \left(H^{\otimes 3} \otimes I_{2^3}\right)|\psi_2\rangle = \frac{1}{2\sqrt{2}}\Big(&\left(H^{\otimes 3}|000\rangle\right)|100\rangle+\left(H^{\otimes 3}|001\rangle\right)|010\rangle+\left(H^{\otimes 3}|010\rangle\right)|000\rangle+\left(H^{\otimes 3}|011\rangle\right)|110\rangle+\\ &+\left(H^{\otimes 3}|100\rangle\right)|000\rangle+\left(H^{\otimes 3}|101\rangle\right)|110\rangle+\left(H^{\otimes 3}|110\rangle\right)|100\rangle+\left(H^{\otimes 3}|111\rangle\right)|010\rangle\Big)\end{aligned}$$

According to (16) $|\psi 3\rangle$ can be written:

$$\begin{aligned}|\psi_3\rangle = \frac{1}{2\sqrt{2}}\Bigg(&\frac{1}{\sqrt{2^3}}\left(|000\rangle+|001\rangle+|010\rangle+|011\rangle+|100\rangle+|101\rangle+|110\rangle+|111\rangle\right)|100\rangle+\frac{1}{\sqrt{2^3}}\left(|000\rangle-|001\rangle+|010\rangle-|011\rangle+|100\rangle-|101\rangle+|110\rangle-\right.\\ &\left.-|111\rangle\right)|010\rangle+\frac{1}{\sqrt{2^3}}\left(|000\rangle+|001\rangle-|010\rangle-|011\rangle+|100\rangle+|101\rangle-|110\rangle-|111\rangle\right)|000\rangle+\frac{1}{\sqrt{2^3}}\left(|000\rangle-|001\rangle-|010\rangle+|011\rangle+|100\rangle-|101\rangle-\right.\\ &\left.-|110\rangle+|111\rangle\right)|110\rangle+\frac{1}{\sqrt{2^3}}\left(|000\rangle+|001\rangle+|010\rangle+|011\rangle-|100\rangle-|101\rangle-|110\rangle-|111\rangle\right)|000\rangle+\frac{1}{\sqrt{2^3}}\left(|000\rangle-|001\rangle+|010\rangle-|011\rangle-|100\rangle+\right.\\ &\left.+|101\rangle-|110\rangle+|111\rangle\right)|110\rangle+\frac{1}{\sqrt{2^3}}\left(|000\rangle+|001\rangle-|010\rangle-|011\rangle-|100\rangle-|101\rangle+|110\rangle+|111\rangle\right)|100\rangle+\frac{1}{\sqrt{2^3}}\left(|000\rangle-|001\rangle-|010\rangle+|011\rangle-\right.\\ &\left.-|100\rangle+|101\rangle+|110\rangle-|111\rangle\right)|010\rangle\Bigg)\end{aligned}$$

$$\begin{aligned}|\psi_3\rangle = \frac{1}{2^3}\big(&|000\rangle|100\rangle+|001\rangle|100\rangle+|010\rangle|100\rangle+|011\rangle|100\rangle+|100\rangle|100\rangle+|101\rangle|100\rangle+|110\rangle|100\rangle+|111\rangle|100\rangle+|000\rangle|010\rangle-|001\rangle|010\rangle+\\ &+|010\rangle|010\rangle-|011\rangle|010\rangle+|100\rangle|010\rangle-|101\rangle|010\rangle+|110\rangle|010\rangle-|111\rangle|010\rangle+|000\rangle|000\rangle+|001\rangle|000\rangle-|010\rangle|000\rangle-|011\rangle|000\rangle+\\ &+|100\rangle|000\rangle+|101\rangle|000\rangle-|110\rangle|000\rangle-|111\rangle|000\rangle+|000\rangle|110\rangle-|001\rangle|110\rangle-|010\rangle|110\rangle+|011\rangle|110\rangle+|100\rangle|110\rangle-|101\rangle|110\rangle-\\ &-|110\rangle|110\rangle+|111\rangle|110\rangle+|000\rangle|000\rangle+|001\rangle|000\rangle+|010\rangle|000\rangle+|011\rangle|000\rangle-|100\rangle|000\rangle-|101\rangle|000\rangle-|110\rangle|000\rangle-|111\rangle|000\rangle+\\ &+|000\rangle|110\rangle-|001\rangle|110\rangle+|010\rangle|110\rangle-|011\rangle|110\rangle-|100\rangle|110\rangle+|101\rangle|110\rangle-|110\rangle|110\rangle+|111\rangle|110\rangle+|000\rangle|100\rangle+|001\rangle|100\rangle-\\ &-|010\rangle|100\rangle-|011\rangle|100\rangle-|100\rangle|100\rangle-|101\rangle|100\rangle+|110\rangle|100\rangle+|111\rangle|100\rangle+|000\rangle|010\rangle-|001\rangle|010\rangle-|010\rangle|010\rangle+|011\rangle|010\rangle-\\ &-|100\rangle|010\rangle+|101\rangle|010\rangle+|110\rangle|010\rangle-|111\rangle|010\rangle\big)\end{aligned}$$

$$\begin{aligned}|\psi_3\rangle = \frac{1}{2^2}\big(&|000\rangle|000\rangle+|000\rangle|010\rangle+|000\rangle|100\rangle+|000\rangle|110\rangle+|001\rangle|000\rangle-|001\rangle|010\rangle+|001\rangle|100\rangle-|001\rangle|110\rangle-\\ &-|110\rangle|000\rangle+|110\rangle|010\rangle+|110\rangle|100\rangle-|110\rangle|110\rangle-|111\rangle|000\rangle-|111\rangle|010\rangle+|111\rangle|100\rangle+|111\rangle|110\rangle\big)\end{aligned}$$

Appendix B

The implementation of classical post-processing part of the algoritm

```
//m1 and m2 are the measured values

//yy[] contains the values of m1 and m2
//represented in base two
print "The system to be solved is: ";
for i=0 to 1 {
  print "s1*",yy[i,0],"+s2*",yy[i,1],"+ s3*",
                    yy[i,2]," = 0";
}
sum[0] = (yy[0,0]+yy[0,1]+yy[0,2]) mod 2;
sum[1] = (yy[1,0]+yy[1,1]+yy[1,2]) mod 2;

p[0] = (yy[0,0]*yy[0,1]*yy[0,2]) mod 2;
p[1] = (yy[1,0]*yy[1,1]*yy[1,2]) mod 2;

i=0; ok=0;
while (i<2) and (ok==0)
{
  if p[i]==0 {
    if sum[i]==1 {ok=1;l=i;}
  }
  i=i+1;
}

if ok==1 {
  for i=0 to 2 {
    if yy[l,i]==1 {k=i;s[k]=0;}
  }
  l = (l+1) mod 2;
  k1=-1; k2=-1;
  for j=0 to 2
  {
    if j!=k {
      if k1<0 {k1=j;}
      else {k2=j;}
    }
  }
  if yy[l,k1]==0 {
    s[k1]=1;
    s[k2]=0;
  }
  else {
    if yy[l,k2]==0 {s[k1]=0; s[k2]=1;}
    else {s[k1]=1; s[k2]=1;}
  }
}
else {
  if (p[0]==0 and p[1]==0){
    for j=0 to 2 { s[j]=1;}
  }
  else {
    if p[0]==0  {l=0;}
    else {l=1;}
    for j=0 to 2
    { s[j]=yy[l,j];}
  }
}
print "s = ",s[0],s[1],s[2];
```