STUDIA INFORMATICA 2002
Volume 23 Number 2A (48)

Jarosław FRANCIK
Politechnika Śląska, Instytut Informatyki

QUANTUM SOFTWARE*

Summary. The paper is an attempt to look at quantum information techniques from the perspective of an software engineer. Focusing on Grover's quantum search, we try to present quantum algorithms using classical engineering methods: pseudocode, high-level programming language, software visualization. It is also analyzed how the specifics of quantum mechanics has to affect these means of description.

OPROGRAMOWANIE KWANTOWE

Streszczenie. Artykuł jest próbą spojrzenia na kwantowe techniki informatyczne z perspektywy inżynierii oprogramowania. Koncentrując się na kwantowym wyszukiwaniu Grovera podjęto próbę przedstawienia algorytmów kwantowych za pomocą klasycznych, informatycznych środków opisu: pseudokodu, języka programowania wysokiego poziomu, wizualizacji oprogramowania. Przeanalizowano, w jaki sposób specyfika mechaniki kwantowej musi wpłynąć na tego typu środki opisu.

1. Introduction

Gershenfeld and Chuang in their popular paper published in *Scientific American* [6] wrote that a hypothetical quantum computer "would look nothing like the machine that sits on your desk; surprisingly, it might resemble the cup of coffee at its side". However splendid as an illustration for dummies, that comparison does not take into account the fact that the essential purpose of both traditional and quantum computing machinery is the same, what could not be said about cups of coffee. The problem with quantum computing is that so far it is mostly a

* The research was supported by KBN Project no. 7 T11C 017 21

domain of physicists rather than computer professionals. The description is made on the level of quantum mechanics and appropriate mathematical model. In this paper we will try to look at the problem as a new branch of informatics. The object of our interests is quantum software: information processing analyzed apart from its physical realization.

Currently quantum computing is a scientific concept rather than a technology, even an emerging one. Compared with traditional solutions, we are not even on the stage of ENIAC. On the other hand we have already started to develop pretty good algorithms. In case of electronic machinery the comparable progress was connected with development of first high-level languages, like Fortran (still most numerical algorithms are traditionally available in this language). These languages are supposed to be essential for any progress in software engineering. Yet modern quantum algorithms are most often designed on the level that is analogous to operation of electronic gates. Consequently, development of high level quantum languages is indispensable. First attempts have already been done [11]. In this paper we will explore a sample quantum algorithm – the Grover's search – using typical tools of a software engineer, keeping in mind all the factors that make the quantum computing so specific.

2. Background

An important demand in quantum computing – resulting from the rules of quantum mechanics – is that every operation must be *reversible*, so that no information may be lost during the computational process [2, 9, 12]. In case an operation is defined by a matrix, such matrix has to be *unitary* ($UU^{\dagger} = U^{\dagger}U = I$, where $U^{\dagger}$ is the conjugate transpose of U). It is really a strong demand, but not a constrain: Lecerf [10] proved that every function computable with a classical Touring machine is also computable with a reversible Touring machine – within polynomial time and memory complexity (running time is in fact within a constant factor from the classical solution). A reversible machine may be in turn directly simulated by a quantum Touring machine – as shown by Benioff [1]. So, every classical problem may be solved on a "quantum platform", what is an important conclusion, however not very surprising.

Speed-up in traditional computing is usually obtained by use of parallelism. Its quantum counterpart is *superposition*. The famous Schrödinger's cat was both dead and alive in the same time: we say that those two states were in superposition. Similarly, a given n-qubit register (a *quregister*) may contain a superposition of all the 2^n possible values. What's more, we can evaluate some function f for all these values, obtaining a superposition of all the results. Problem is that once the box is opened, the superposition is gone: the cat is just alive *or* dead (there is nothing mystical about his experience!), and so is the quregister content: once the measurement is done, just a single, randomly chosen computation result is available. In order to get all values of f this way, the

function should be called exponential number of times, what makes no gain over traditional solutions.

What makes the quantum computing really powerful is the *interference*. Thanks to superposition, we can deal with all the function results simultaneously, each result represented by a quantum state. To make use of interference, the focus is moved from direct computation of function rather to *transforming the probability distribution* over various states. To precisely describe it, instead of probability, some complex values called *probability amplitudes* are used for each state. The probability in any state is given by the square of the absolute value of the amplitude in this state (so, the sum of squares of all the amplitudes must be equal to 1). The key point of quantum computation is transforming the amplitude vector. Performing a single transformation means that all the amplitudes change simultaneously and, what's more, all interfere – a new value of each amplitude depends on the values of all the other amplitudes. From the computational point of view it makes calculating n functions of n variables each in a single step, what may be shown as using a matrix operator O:

$$A_{n+1} = O \cdot A_n \tag{1}$$

where A_n, A_{n+1} are amplitude state vectors, and O is unitary.

The goal of quantum computing is to process – by clever use of interference – the amplitude vector so that to gain the probability of the desired state. The act of *measurement* of this state finishes a quantum algorithm.

3. Grover's Search Algorithm: a computational view

In previous section *superposition* and *interference* of quantum states have been pointed as the essential factors for obtaining the "quantum speed-up". Grover's search algorithm [3, 7, 8] makes great use of these phenomena. The complexity of classical search of an unordered collection of N items is of $O(N)$. Using the quantum algorithm we can reduce this factor to $O(\sqrt{N})$.

The algorithm uses a single quregister of 2^n qubits, where $2^n \geq N$. For simplicity assume that N is a power of 2 and $2^n = N$. The outline is as follows:

1. In first step a superposition of all possible 2^n states is generated.
2. A transformation is performed that makes the probability amplitude of the desired state differ from the other amplitudes (*quantum oracle*).
3. Another transformation (*diffusion*) is performed that gains the probability of this state.
4. Steps 2. and 3. are iterated as many times as needed.
5. Algorithm terminates when the probability of the desired state is close to 1. Then the measurement is performed.

Superposition. This is a common operation performed with standard quantum *Hadamard operator*, which results in giving the same value to all the possibility amplitudes. In fig. 1a this situation is depicted for a 4-qubit register, that contains different 16 values. As the sum of squares of all the amplitudes is 1, the value of each amplitude a_i becomes as follows:

$$\forall i \in (0..N-1) \quad a_i := \frac{1}{\sqrt{N}} = \frac{1}{\sqrt{2^n}} = 2^{-n/2} \tag{2}$$

Quantum oracle, the second step of the algorithm, is the point in which the searched item is identified. The *oracle* function decides if its argument is the item being searched or not, and it is simultaneously called for each item of the scanned collection. The operation has to be reversible, so no information may be lost. This is obtained by simple negating the amplitude of the searched item (fig. 1b). Given that x is the state being searched we get:

$$\forall i \in (0..N-1) \quad a_i := (i == x)\ ?\ -a_i : a_i \tag{3}$$

In the formula above we borrow some standard operator notation from the language C.

As the amplitude is complex, the negating operation is described rather as rotation of the marked state by a phase of π. Anyhow, this operation would not change the possibility of detecting this state if a measurement was done at this point.

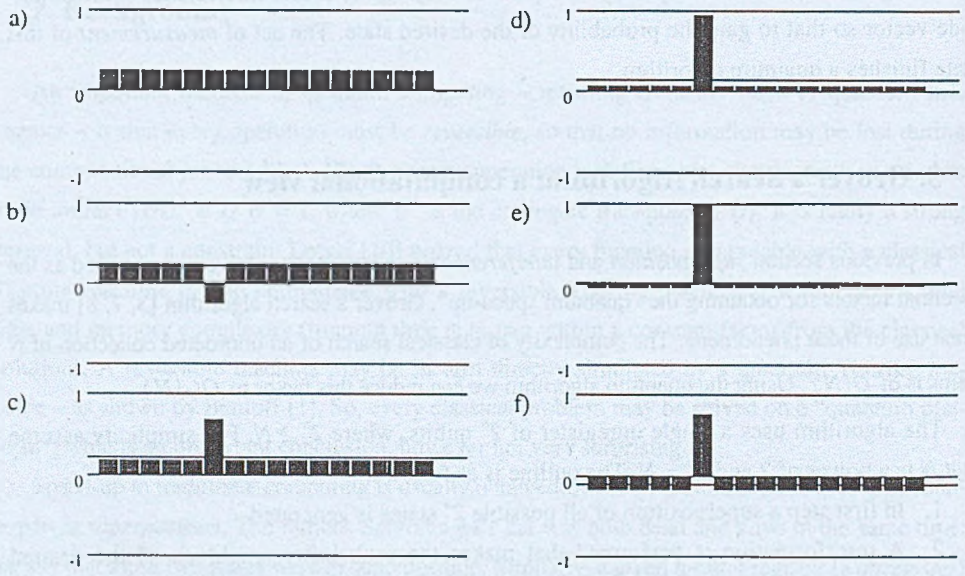

Fig. 1. Visualization of the Grover's algorithm
Rys. 1. Wizualizacja algorytmu Grovera

Diffusion operator, the algorithm's next step, is where the use of interference is made. This operator is often referred as the *inversion about average*, as, indeed, it is given as follows:

$$\forall i \in (0..N-1) \quad a_i := -a_i + 2\bar{a} = -a_i + \frac{2}{N}\sum_{j=0}^{N-1} a_j \tag{4}$$

where $\bar{a}$ is the average of all the amplitude values. Notice, that this operation is applied on a vector in which all components, except one, are equal to a value, say α, the one component that is different is negative (fig. 1b). The average $\bar{a}$ is also approximately equal to α, so N - 1 components do not change significantly as a result of the inversion about average. The one component that was negative becomes positive and increases by approximately 2α (fig. 1c).

Iteration. The gain effect obtained by a single inversion about average is not enough, especially if to notice that the more states we have the less is the gain. After some number of iterations the possibility amplitude becomes close to 1. It is almost never equal to 1 – it results from the fact, that Grover's algorithm, as most quantum algorithms, is a *probabilistic algorithm*: the proper result is obtained with high probability, but is not guaranteed. Consecutive iterations are shown in fig. 1c – 1f. At the 3rd iteration (fig. 1e) the amplitude of the desired state has its maximum (for case of 16 states); afterwards it decreases (fig. 1f). This result is surprising only at first sight. The value of the $N - 1$ components, denoted α, slowly decreases; once it exceeds zero the gain effect reverses. In fact both values are quantified values of some periodical function (the desired amplitude grows again after iteration no 9, 15 and so on).

The problem is: how many times should we iterate to get the optimal result? Let α denote, as above, the value of all the amplitudes but one, and β is the amplitude of the state being searched a_x. Then the average is given as:

$$\bar{a} = \frac{1}{N}((N-1)\alpha - \beta) \tag{5}$$

Using (4), (3) and (2) it holds:

$$\begin{aligned} \alpha_{k+1} &= -\alpha_k + \frac{2}{N}((N-1)\alpha_k - \beta_k) = (1-\frac{2}{N})\alpha_k - \frac{2}{N}\beta_k \\ \beta_{k+1} &= -(-\beta_k) + \frac{2}{N}((N-1)\alpha_k - \beta_k) = (1-\frac{2}{N})\beta_k + \frac{2(N-1)}{N}\alpha_k \\ \alpha_0 &= \beta_0 = 2^{-n/2} \end{aligned} \tag{6}$$

where α_k and β_k are previous states, and α_{k+1} and β_{k+1} are the next states of α and β.

The difference equation (6) leads to a characteristic square equation with complex roots. The analytic approach may be tricky, but one can notice that after a substitution $\chi = \alpha\sqrt{N-1}$ the equation (6) becomes an equation of rotation a point (χ, β) around the center of the coordinate system by an angle Δx, $\sin\Delta x = 2\sqrt{N-1}/N$. The best result is achieved when the angle reaches $\pi/2$, so the number of iterations is:

$$k_{max} = \frac{\pi/2}{\Delta x} \approx \frac{\pi/2}{2\sqrt{N-1}/N} \approx \frac{\pi\sqrt{N}}{4} \tag{7}$$

After the last iteration the final measurement is performed.

4. Presentation

In the previous section the Grover's algorithm has been defined by observing just what happens with individual values of probability amplitude vector. One can notice that some elements of mathematical description that are common for most papers on quantum algorithmics have been omitted (for example tensor notation). The probability amplitudes are treated just as program variables. This makes a temptation to notify the algorithm using something similar to high-level programming language.

The sample code presented in fig. 2 uses a pseudo-code formalism very similar to C language, and thus called *quC*. It introduces *quregisters* along with a special notation *A<i>* that denotes the probability amplitude of the i-th state of *A* (triangle bracket being a symbol of state index). Special constructions have been also proposed to express quantum parallelism. A new statement *qufor* have been added for a range of operations simultaneously executed on multiple quantum states. Another new block instruction, *qusum*, allows instantaneous summarizing expressions, which is indispensable when modeling the interference. The *qusum* instruction may appear in expressions in place of a function call.

```
int qufunction Grover(int x)
{ const N = 16;
  quregister A[sqrt(N)];
  int i, j;

  qufor (<i>=<0..N-1>) A<i> = 1 / sqrt(N);     // superposition
  for (j = 0; j < pi*sqrt(N)/4; j++)
  { qufor (<i>=<0..N-1>) A<i> = (i == x) ? -A<i> : A<i>;
    qufor (<i>=<0..N-1>) A<i> = -A<i> + 2 *
      qusum(i = 0 to N-1) { return A<i>;}
  }
  return measure(A);
}
```

Fig. 2. Grover's algorithm in quC pseudo-code
Rys. 2. Algorytm Grovera zapisany w pseudokodzie quC

The quC pseudo-code resembles a C program and may be easily converted – one has just to unbound *qufor/qusum* statements and replace quregister amplitudes with floated-point arrays. Such conversion has been done, and the resulted program has been visualized using a Daphnis algorithm animation system [5]. The images created by this tool have been presented in fig. 1, that was applied as an illustration in the previous section.

5. Discussion and Conclusion

A pseudo-code notation as depicted in fig. 2 makes a good explanatory tool for informaticians who want to understand the quantum software, foremost because it is familiar in form. The weak point of this approach is that it is not consistent with the rules of quantum mechanics. It lacks a system of coherent, internal verification of such properties as reversibility, unitarity and normalization. These properties are crucial in terms of computability of quantum software, and should be incorporated into a quantum programming language as its integral part. It is probably unavoidable in any language based on applying open mathematical formulas. What's more, such a language could be extremely difficult in phase of automatic translating the source code into signals controlling a circuit of quantum gates.

Another possibility is a language based on a matrix notation of operations. All the operations performed by any quantum algorithm may be expressed as matrix operations – like (1). The matrices of Grover's algorithm operations may be found in [3, 7, 8]. Once a matrix is given it's quite comfortable; an engineer may reconstruct all the operations performed; it is also easy to check if an operation is reversible. Much more difficult is to propose a proper matrix representation when implementing a conceptually new algorithm.

Most languages existing so far, as the QCL [120], base on a predefined set of ready-to-use quantum transformations, that the underlying equipment is supposed to support. This approach makes programs strongly tied to the hardware, and although quite useful, it may resemble an assembler language rather then a high-level one.

Real quantum programming will not be just rewriting the Grover's algorithm. A practical need will emerge for not very sophisticated routines that will have to be implemented quantum. A mine of examples is the query function for the Grover's search oracle: whatever would be implemented, must be called in superposition, so must be quantum. And this is where a practical, easy-to-use high-level quantum language will be essential.

Maybe the practical approach of the future will be rather creating tools for machine translation of classical algorithms into an assembly-level quantum specification.

BIBLIOGRAPHY

1. Benioff P. A.: Quantum mechanical Hamiltonian models of Touring machines. Journal of Statistical Physics, 29(3), 1982, pp. 515-546.
2. Bugajski S., Klamka J., Węgrzyn S.: Foundations of Quantum Computing. Part I. Archiwum Informatyki Teoretycznej i Stosowanej, Vol. 13 No 2/2001, pp. 96-142.

3. Bugajski S.: Quantum Serach. Archiwum Informatyki Teoretycznej i Stosowanej, Vol. 13 No 2/2001, pp. 143-150.
4. Colwel B.: Engineering, Science and Quantum Mechanics, IEEE Comp. 35/2002, pp. 8 – 10.
5. Francik J.: Algorithm Animation Using Data Flow Tracing. In: S. Diehl (Ed.): Software Visualization. LNCS 2269, Springer Verlag, 2002, pp. 73-87.
6. Gershenfeld N., Chuang I. L.: Quantum Computing with Molecules, Scientific American, June 1998 (reprinted in Polish in Świat Nauki No 8/1998).
7. Grover L. K.: A Fast Quantum Mechanical Algorithm for Database Search. Proc of STOC 1996, Philadelphia PA (ACM Press New York), pp. 212-219.
8. Klamka J.: Quantum search algorithm. Seminarium Sieci Komputerowe, Zakopane 2002.
9. Koskela J.-P., Mettinen K.: Why do quantum algorithms work? (Deutsch-Josza problem). Quantum Computing, Dept. of Comp. Science, Univ. of Helsinki 1998.
10. Lecerf Y.: Machines de Touring réversibles. Récursive insolubilité en $n \in N$ de l'équation $u = \theta^n$ où θ est un isomorphisme de codes. Comptes rendus de l'Académie française des sciences, 257, 1963, pp. 2597-2600.
11. Ömer B.: Quantum Programming in QCL. PhD thesis. Institute of Inf. Systems, Technical University of Vienna, Austria 2000.
12. Węgrzyn A., Klamka J.: Quantum Systems of Informatics, IITiS PAN, Gliwice 2000.

Recenzent: Dr inż. Ryszard Winiarczyk

Wpłynęło do Redakcji 30 kwietnia 2002 r.

Streszczenie

Problem z informatyką kwantową polega na tym, że ciągle jest to jeszcze bardziej domena fizyków niż informatyków. Artykuł jest próbą spojrzenia na tę nową dziedzinę okiem inżyniera – programisty. Przedstawiono – w dużym skrócie – najistotniejsze pojęcia mające wpływ na kształt algorytmów kwantowych: odwracalność, superpozycję, interferencję. Koncentrując się na kwantowym wyszukiwaniu Grovera podjęto próbę przedstawienia algorytmów kwantowych za pomocą klasycznych, informatycznych środków opisu. Algorytm został poddany prostej analizie numerycznej. Przedstawiono jego prostą wizualizację (rys. 1) oraz zapis w pseudokodzie (rys. 2). Przeanalizowano przydatność i ograniczenia takich środków opisu, zasygnalizowano też istnienie innych narzędzi, w tym języków programowania (np. QCL 11).