ISSN No. 0976-5697

Volume 2, No. 4, July-August 2011

International Journal of Advanced Research in Computer Science

RESEARCH PAPER

Available Online at www.ijarcs.info

Analysis of Quantum Algorithm to Find the Solution of Integer factorization Problem

Anil Kumar Malviya*
Computer Science & Engineering Department,
K.N.I.T., Sultanpur, U.P., India
anilkmalviya@yahoo.com

Vibhooti Singh
Computer Science &Engg. Department
Mewar University, Rajsthan, India
vibhootisingh@gmail.com

Lokendra Singh Umrao
Computer Science &Engineering Department,
F.G.I.E.T., Raebareli, U.P., India
lokendramanit@gmail.com

Abstract: - Quantum computing has emerged as an important interdisciplinary field, merging theories in mathematics, physics and computer science. So far, a significant portion of research in quantum computing has focused on the design of quantum algorithms. Quantum Computers requires very different algorithms for factorization, which plays an important role in the field of quantum computing and shor's algorithm is used to address integer factorization problem. Shor's algorithm is important because it breaks a widely used public-key cryptography scheme known as RSA. This paper discusses the refinement of quantum factorization algorithm.

***Keywords*:** - quantum algorithm, factorization, gcd, prime decomposition, shor's algorithm.

I. INTRODUCTION

Quantum computer first proposed in the 1970s,[1] quantum computing relies on quantum physics by taking advantage of certain quantum physics properties of atoms or nuclei that allow them to work together as quantum bits, or qubits, to be the computer's processor and memory. By interacting with each other while being isolated from the external environment, qubits can perform certain calculations exponentially faster than conventional computers.A quantum computer is a machine that performs calculations based on the laws of quantum mechanics, which is the behavior of particles at the sub-atomic level. By the early nineties it was know that a quantum computer could be faster than any classical computer for certain problems.

In a quantum computer, the fundamental unit of information (called a quantum bit or qubit), is not binary but rather more quaternary in nature. This qubit property arises as a direct consequence of its adherence to the laws of quantum mechanics which differ radically from the laws of classical physics [2]. A qubit can exist not only in a state corresponding to the logical state 0 or 1 as in a classical bit, but also in states corresponding to a blend or superposition of these classical states. In other words, a qubit can exist as a zero, a one, or simultaneously as both 0 and 1, with a numerical coefficient representing the probability for each state.

Integer factorization is believed to be Computationally infeasible with an ordinary computer for large integers that are the product of only a few prime numbers (e.g., products of two 300-digit primes). By comparison, a quantum computer could solve this problem more efficiently than a classical computer using Shor's algorithm to find its factors. This ability would allow a quantum computer to "break" many of the cryptographic systems in use today, in the sense that there would be a polynomial time (in the number of bits of the integer) algorithm for solving the problem. In particular, most of the popular public key ciphers are based on the difficulty of factoring integers, including forms of RSA. These are used to protect secure Web pages, encrypted email, and many other types of data. Breaking these would have significant ramifications for electronic privacy and security. The only way to increase the security of an algorithm like RSA would be to increase the key size and hope that an adversary does not have the resources to build and use a powerful enough quantum computer[3].

II. RELATED WORK

The most famous example of the high performance of a quantum computer is Peter Shor's algorithm for factoring large numbers. The solution of this problem is important in cryptography. Actually Shor solved a related problem called the discrete log. Suppose we take a number x to the power r and reduce the answer modulo *n* (i.e., find the remainder r after dividing x^r by n). This is straightforward to calculate. It is much more difficult to find the inverse - given x, *n*, and y, find r such that $x^r = y \pmod{n}$. For factoring, all we need to do is consider y=1 and find the smallest positive r such that $x^r = 1 \pmod{n}$. Shor's quantum algorithm to do this calculates x^r for all r at once. Since $x^{l+r} = x^l \pmod{n}$, this is a periodic function with period r. Then when we take the Fourier transform, we will get something that is peaked at multiples of 1/r. Since there is an efficient quantum algorithm for the Fourier transform, we can therefore find r.

A. Integer Factorization

a. Prime decomposition
b. Practical applications

In number theory, the integer factorization problem is the problem of finding a non-trivial divisor of a composite number; for example, given a number like 91, the challenge is to find a number such as 7 which divides it. When the numbers are very large, no efficient algorithm is known; a recent effort which factored a 200 digit number (RSA-200) took eighteen months and used over half a century of computer time. The supposed difficulty of this problem is at the heart of certain algorithms in cryptography such as RSA. Many areas of mathematics and computer science have been brought to bear on the problem, including Elliptic curves, algebraic number theory, and quantum computing.

B. Prime decomposition

By the fundamental theorem of arithmetic, every integer has a unique prime factorization. Given an algorithm for integer factorization, one can factor any integer down to its constituent primes by repeated application of this algorithm [5].

C. Practical applications

The hardness of this problem is at the heart of several important cryptographic systems. A fast integer factorization algorithm would mean that the RSA public-key algorithm was insecure. Some cryptographic systems, such as the Rabin public- key algorithm and the Blum Shub pseudo- random number generator can make a stronger guarantee - any means of breaking them can be used to build a fast integer factorization algorithm, so if integer factorization is hard then they are strong. In contrast, it may turn out that there are attacks on the RSA problem more efficient than integer factorization, though none are currently known.

If a large, n-bit number is the product of two primes that are roughly the same size, then no algorithm is known that can factor in polynomial time. That means there is no known algorithm that can factor it in time O (n^k) for any constant *k*. There are algorithms, however, that are faster than O (e^n). In other words, the best known algorithms are sub- exponential, but super-polynomial. In particular, the best known asymptotic running time is for the general number field sieve (GNFS) algorithm, which is:

$$\Theta\left(\exp\left(\left(\frac{64}{9}n\right)^{\frac{1}{3}}(\log n)^{\frac{2}{3}}\right)\right).$$

For an ordinary computer, GNFS is the best known algorithm for large *n*. For a quantum computer, however, Peter Shor discovered an algorithm in 1994 that solves it in polynomial time. This will have significant implications for cryptography if a large quantum computer is ever built. Shor's algorithm takes only O (($log\ n$)^3) time and O($log\ n$) space. In 2001, the first 7-qubit quantum computer became the first to Shor's algorithm. It factored the number 15.

III. ANALYSIS OF SHOR'S QUANTUM FACTORIZATION ALGORITHM

a. Procedure
   [i] Classical part
   [ii] Quantum part: Period-finding subroutine:
b. Explanation of the algorithm
   [i] Obtaining factors from period b) Finding the period
c. Modifications to Shor's Algorithm

Shor's algorithm is a quantum algorithm for factoring a number *n* in O ((log *n*) ^{3}) time and O (log *n*) space, named after Peter Shor[3].

The algorithm is significant because it implies that public key cryptography might be easily broken, given a sufficiently large quantum computer. RSA, for example, uses a public key *n* which is the product of two large prime numbers. One way to crack RSA encryption is by factoring *n*, but with classical algorithms, factoring becomes increasingly time-consuming as *n* grows large; more specifically, no classical algorithm is known that can factor in time O((log $n)^k$) for any *k*. By contrast, Shor's algorithm can crack RSA in polynomial time. It has also been extended to attack many other public key cryptosystems.

Like all quantum computer algorithms, Shor's algorithm is probabilistic: it gives the correct answer with high probability, and the probability of failure can be decreased by repeating the algorithm. Shor's algorithm was demonstrated in 2001 by a group at IBM, which factored 15 into 3 and 5, using a quantum computer with 7 qubits.Like most factorization algorithms, Shor's algorithm reduces the factorization[3].

Problem to problem of finding the period of a function uses quantum parallelism to find a superposition of all values of the function in one step.

Then it calculated the QFT of the function, which sets the amplitudes into multiples of the fundamental.

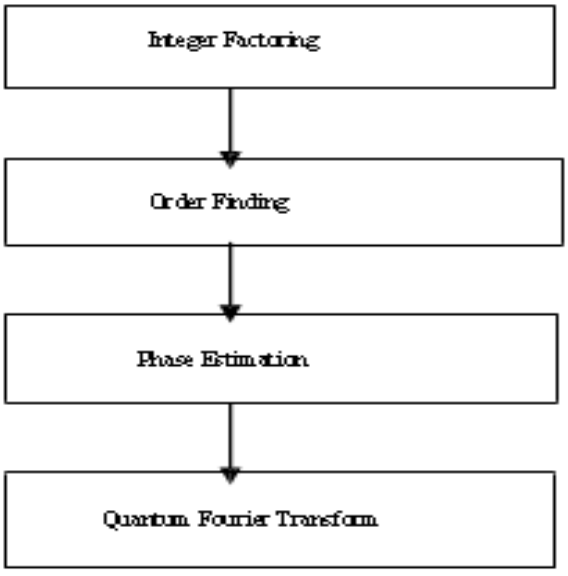

Figure-1: Steps of Shor's algorithm at a glance.

A. Procedure

He problem we are trying to solve is that, given an integer *n*, we try to find another integer *p* between *1* and *n* that divides *n*.

Shor's algorithm consists of two parts:

i. A reduction of the factoring problem to the prob-

lem of order-finding, which can be done on a classical computer.

ii. A quantum algorithm to solve the order finding problem.

a. Classical Part

Pick a pseudo-random number $a < n$

Compute gcd (a, n). This may be done using the Euclidean algorithm.

If gcd (a, n) $\neq 1$, then there is a nontrivial factor of n, so we are done.

Otherwise, use the period-finding subroutine (below) to find r, the period of the following function:

f (x)=a^x mod n,

i.e. the smallest integer r for which $f(x + r) = f(x)$. If r is odd, go back to step 1.

If $a^{r/2} \equiv -1 \pmod{n}$, go back to step 1.

gcd ($a^{r/2} \pm 1$, n) is a nontrivial factor of n. We are done.

b. Quantum Part: Period-Finding Subroutine

The quantum circuits used for this algorithm are custom designed for each choice of n and the random a used in $f(x) = a^x \bmod n$. Given n, find $Q = 2^q$ such that $n^2 \leq Q < 2n^2$, which implies $Q / r > n$. The input and output qubit registers need to hold superpositions of values from 0 to $Q - 1$, and so have q qubits each. Using what might appear to be twice as many qubits as necessary guarantees that there is at least n different x which produce the same $f(x)$, even as the period r approaches $n/2$.

B. Explanation of the Algorithm

The algorithm is composed of two parts. The first part of the algorithm turns the factoring problem into the problem of finding the period of a function, and may be implemented classically. The second part finds the period using the quantum Fourier transform, and is responsible for the quantum speedup.

a. Obtaining factors from period

The integers less than n and coprime with n form a finite group under multiplication modulo n. By the end of step 3, we have an integer a in this group. Since the group is finite, a must have a finite order r, the smallest positive integer such that

$a^r \equiv 1 \bmod n$.

Therefore, $n \mid (a^r - 1)$. Suppose we are able to obtain r, and it is even. Then

$a^r - 1 = (a^{r/2} - 1)(a^{r/2} + 1) \equiv 0 \bmod n$

$n \mid (a^{r/2} - 1)(a^{r/2} + 1)$.

r is the *smallest* positive integer such that $a^r \equiv 1$, so n cannot divide $(a^{r/2} - 1)$. If n also does not divide $(a^{r/2} + 1)$, then n must have a nontrivial common factor with each of $(a^{r/2} - 1)$ and $(a^{r/2} + 1)$.

Proof: For simplicity, denote $(a^{r/2} - 1)$ and $(a^{r/2} + 1)$ by u and v respectively. $n \mid uv$, so $kn = uv$ for some integer k. Suppose gcd(u, n) = 1; then $mu + ln = 1$ for some integers m and l (this is a property of the greatest_common_divisor.) Multiplying both sides by v, we find that $mkn + nvl = v$, so $n \mid v$. By contradiction, gcd(u, n) $\neq 1$. By a similar argument, gcd(v, n) $\neq 1$.

This supplies us with a factorization of n. If n is the product of two primes, this is the only possible factorization.

b. Finding the period

Shor's period-finding algorithm relies heavily on the ability of a quantum computer to be in many states simultaneously. Physicists call this behavior a "superposition" of states. To compute the period of a function f, we evaluate the function at all points simultaneously[4].

Quantum physics does not allow us to access all this information directly, though. A measurement will yield only one of all possible values, destroying all others. But for the no cloning theorem, we could first measure $f(x)$ without measuring x, and then make a few copies of the resulting state (which is a superposition of states all having the same $f(x)$). Measuring x on these states would provide different x values which give the same $f(x)$, leading to the period. Because we cannot make exact copies of a quantum state, this method does not work. Therefore we have to carefully transform the superposition to another state that will return the correct answer with high probability. This is achieved by the quantum Fourier transform [9].

Shor thus had to solve three "implementation" problems. All of them had to be implemented "fast", which means that they can be implemented with a number of quantum gates that is polynomial in log n.

a. Create a superposition of states. This can be done by applying Hadamard gates to all qubits in the input register. Another approach would be to use the quantum Fourier transform (see below).

b. Implement the function f as a quantum transform. To achieve this, Shor used repeated squaring forhis modular exponentiation transformation. It is important to note that this step is more difficult to implement than the quantum Fourier transform, in that it requires ancillary qubits and substantially more gates to accomplish.

c. Perform a quantum Fourier transform. By using controlled rotation gates and Hadamard gates Shor designed a circuit for the quantum Fourier transform (with $Q = 2^q$) that uses just $q(q - 1) / 2 = O((\log Q)^2)$ gates.

After all these transformations a measurement will yield an approximation to the period r. For simplicity assume that there is a y such that yr/Q is an integer. Then the probability to measure y is 1. To see that we notice that then

$$e^{-2\pi i byr / Q} = 1$$

for all integers b. Therefore the sum whose square gives us the probability to measure y will be Q/r since b takes roughly Q/r values and thus the probability is $1 / r^2$. There are r y such that yr/Q is an integer and also r possibilities for $f(x_0)$, so the probabilities sum to 1.

Note: another way to explain Shor's algorithm is by noting that it is just the quantum phase estimation algorithm in disguise.

C. *Modifications to Shor's Algorithm*

There have been many modifications to Shor's algorithm. For example, whereas, an order of twenty to thirty runs are required on a quantum computer in the case of Shor's original algorithm, if the period of the series ends up being odd.

IV. CONCLUSION

In this paper, we present a quantum factorization algorithm. The algorithm's probability of success is higher than shor's algorithm. As we know, up to date there isn't a quantum algorithm for NPC problem in polynomial time. So finding a quantum algorithm for NPC needs more research.

V. REFERENCES

[1] Hoi-Kwong Lo, Tim Spiller, Sandu Popescu- "Introduction to Quantum Computationand Information"-1st edition, World Scientific.

[2] P.W.Shor, "Polynomial time algorithms for prime factorization and discrete logarithms on a quantum computer".

[3] P.W.Shor, "Why haven't more quantum algorithms beefound?" J. ACM 50, 87-90(2003).Siam J.Comput.26, pp.1484-1509(1997).

[4] Shor,P.W.(1994) "Algorithms for Quantum Computation: Discrete Logarithms and factoring" 35th annual Symposium on Foundations of Computer Science, IEEE.pp.124-134.

[5] Grover, Lov, Quantum computing, published in The Sciences, July/August 1999, pp 24-30.

[6] Scarani, Valerio, Quantum computing, American Journal of Physics, Vol. 66, 1998, pp956-960.

[7] Steane, Andrew, Quantum Computing, Quantum Physics, Vol. 61, (1998), pp117-173.

[8] Vedral, Vlatko, and Martin B. Plenio, Basics in quantum computation, Progress in Quantum Electronics, 1998, pp1-39.

[9] Barenco, Andriano, Quantum Physics and Computers, Contemporary Physics, Vol.37, 1996, pp375-389.

[10] Brandt, Howard E., Qubit devices and the issue ofquantum decoherence, Progress in Quantum Electronics, Vol. 22, (1999).

[11] Deutsch, David, Quantum theory, the Church- Turing principle and the universal quantum computer, Proc. R. Soc. Lond. A400, 97-117 (1985).

[12] M.A.Nielsen and L.L.Chuang. "Quantum Computation and Quantum Information". Cambridge University, 2000.