Quantum networks for elementary arithmetic operations

Vlatko Vedral,[*] Adriano Barenco, and Artur Ekert
Clarendon Laboratory, Department of Physics, University of Oxford, Oxford OX1 3PU, United Kingdom
(Received 3 November 1995)

Quantum computers require quantum arithmetic. We provide an explicit construction of quantum networks effecting basic arithmetic operations: from addition to modular exponentiation. Quantum modular exponentiation seems to be the most difficult (time and space consuming) part of Shor's quantum factorizing algorithm. We show that the auxiliary memory required to perform this operation in a reversible way grows linearly with the size of the number to be factorized. [S1050-2947(96)05707-1]

PACS number(s): 03.65.Ca, 07.05.Bx, 89.80.+h

I. INTRODUCTION

A quantum computer is a physical machine that can accept input states which represent a coherent superposition of many different possible inputs and subsequently evolve them into a corresponding superposition of outputs. Computation, i.e., a sequence of unitary transformations, affects simultaneously each element of the superposition, generating a massive parallel data processing albeit within one piece of quantum hardware [1]. This way quantum computers can efficiently solve some problems which are believed to be intractable on any classical computer [2,3]. Apart from changing the complexity classes, the quantum theory of computation reveals the fundamental connections between the laws of physics and the nature of computation and mathematics [4].

For the purpose of this paper a quantum computer will be viewed as a quantum network (or a family of quantum networks) composed of quantum logic gates, each gate performing an elementary unitary operation on one, two, or more two-state quantum systems called *qubits* [5]. Each qubit represents an elementary unit of information; it has a chosen "computational" basis $\{|0\rangle,|1\rangle\}$ corresponding to the classical bit values 0 and 1. Boolean operations which map sequences of 0's and 1's into other sequences of 0's and 1's are defined with respect to this computational basis.

Any unitary operation is reversible. That is why quantum networks effecting elementary arithmetic operations such as addition, multiplication, and exponentiation cannot be directly deduced from their classical Boolean counterparts [classical logic gates such as AND or OR are clearly irreversible: reading 1 at the output of the OR gate does not provide enough information to determine the input which could be either (0,1) or (1,0) or (1,1)]. Quantum arithmetic must be built from reversible logical components. It has been shown that reversible networks (a prerequisite for quantum computation) require some additional memory for storing intermediate results [6,7]. Hence the art of building quantum networks is often reduced to minimizing this auxiliary memory or to optimizing the tradeoff between the auxiliary memory and a number of computational steps required to complete a given operation in a reversible way. In many situations, it is possible and even advisable to perform certain parts of a quantum computation in a classical way, using conventional irreversible computers. This is because classical storage and irreversible manipulation of information is much easier than its quantum storage and reversible coherent computation. In particular, this can be realized in computing constants and parameters which do not require to be placed into a coherent superposition of different values. Hence there is usually a practical distinction between quantum variables and classical parameters which frequently undergo different treatments in the same quantum computation. An example of this can be seen in the forthcoming section, and is extensively used throughout this work.

In this paper we provide an explicit construction of several elementary quantum networks. We focus on the space complexity, i.e., on the optimal use of the auxiliary memory. In our constructions, we save memory by reversing some computations with different computations (rather than with the same computation but run backwards, as it is the case in the so-called "pebble game" [7]). The networks are presented in the ascending order of complication. We start from a simple quantum addition, and end up with a modular exponentiation

$$U_{a,N}|x\rangle\otimes|0\rangle\rightarrow|x\rangle\otimes|a^x \bmod N\rangle, \tag{1}$$

where a and N are predetermined and known parameters. This particular operation plays an important role in Shor's quantum factoring algorithm [3] and seems to be its most demanding part.

The structure of the paper is as follows: in Sec. II we define some basic terms and describe methods of reversing some types of computation, in Sec. III we provide a detailed description of the selected quantum networks, and in Sec. IV we discuss their complexity.

II. BASIC CONCEPTS

For completeness let us start with some basic definitions. A quantum network is a quantum computing device consisting of quantum logic gates whose computational steps are synchronized in time. The outputs of some of the gates are connected by wires to the inputs of others. The size of the network is its number of gates. The size of the input of the

[*]Present address: Blackett Laboratory, Imperial College, Prince Consort Road, London SW7 2BZ, United Kingdom.

network is its number of input qubits, i.e., the qubits that are prepared appropriately at the beginning of each computation performed by the network. Inputs are encoded in binary form in the computational basis of selected qubits often called a *quantum register*, or simply a *register*. For instance, the binary form of number 6 is 110 and loading a quantum register with this value is done by preparing three qubits in state $|1\rangle \otimes |1\rangle \otimes |0\rangle$. In the following we use a more compact notation: $|a\rangle$ stands for the direct product $|a_n\rangle \otimes |a_{n-1}\rangle \cdots |a_1\rangle \otimes |a_0\rangle$ which denotes a quantum register prepared with the value $a = 2^0 a_0 + 2^1 a_1 + \cdots + 2^n a_n$. Computation is defined as a unitary evolution of the network which takes its initial state ''input'' into some final state ''output.''

Both the input and the output can be encoded in several registers. Even when f is a one-to-one map between the input x and the output $f(x)$ and the operation can be formally written as a unitary operator U_f,

$$U_f |x\rangle \rightarrow |f(x)\rangle, \tag{2}$$

we may still need an auxiliary register to store the intermediate data. When f is not a bijection we have to use an additional register in order to guarantee the unitarity of computation. In this case the computation must be viewed as a unitary transformation U_f of (at least) two registers,

$$U_f |x,0\rangle \rightarrow |x, f(x)\rangle, \tag{3}$$

where the second register is of appropriate size to accommodate $f(x)$.

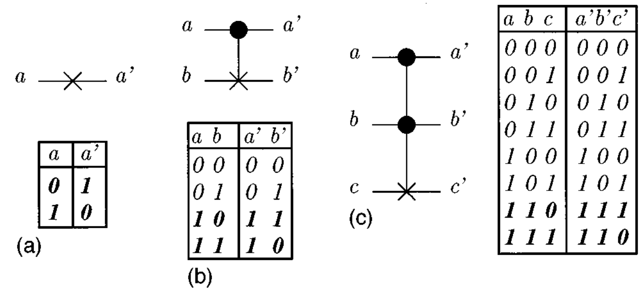
As an example, consider a function $f_{a,N}: x \rightarrow ax \bmod N$. A quantum network that effects this computation takes the value x from a register and multiplies it by a parameter a modulo another parameter N. If a and N are coprime, the function is bijective in the interval $\{0,1,\ldots,N-1\}$, and it is possible to construct a network that writes the answer into the same register which initially contained the input x [as in Eq. (2)]. This can be achieved by introducing an auxiliary register and performing

$$U_{a,N} |x,0\rangle \rightarrow |x, ax \bmod N\rangle. \tag{4}$$

Then we can precompute $a^{-1} \bmod N$, the inverse of a modulo N (this can be done classically in an efficient way using Euclid's algorithm [8]), and, by exchanging the two registers and applying $U^{-1}_{a^{-1}\bmod N, N}$ to the resulting state, we obtain

$$U^{-1}_{a^{-1}\bmod N,N} S |x, ax \bmod N\rangle \rightarrow U^{-1}_{a^{-1}\bmod N,N} |ax \bmod N, x\rangle$$
$$\rightarrow |ax \bmod N, 0\rangle, \tag{5}$$

where S is a unitary operation that exchanges the states of the two registers. Note that $U^{-1}_{a^{-1}\bmod N,N}$ is the unitary transformation representing the inverse of modular multiplication by the inverse of a. In general, given a network of elementary gates implementing a unitary operation U, the inverse of U can be realized by a network of equal size consisting of the inverses of these elementary gates taken in the reverse order. Thus,



a	a'
0	1
1	0

a b	a' b'
0 0	0 0
0 1	0 1
1 0	1 1
1 1	1 0

a b c	a'b'c'
0 0 0	0 0 0
0 0 1	0 0 1
0 1 0	0 1 0
0 1 1	0 1 1
1 0 0	1 0 0
1 0 1	1 0 1
1 1 0	1 1 1
1 1 1	1 1 0

FIG. 1. Truth tables and graphical representations of the elementary quantum gates used for the construction of more complicated quantum networks. The control qubits are graphically represented by a dot, the target qubits by a cross. (a) NOT operation. (b) control-NOT. This gate can be seen as a ''copy operation'' in the sense that a target qubit (b) initially in the state 0 will be after the action of the gate in the same state as the control qubit. (c) Toffoli gate. This gate can also be seen as a control-control-NOT: the target bit (c) undergoes a NOT operation only when the two controls (a and b) are in state 1.

$$U^{-1}_{a^{-1}\bmod N,N} S U_{a,N} |x,0\rangle \rightarrow |ax \bmod N, 0\rangle \tag{6}$$

effectively performs

$$|x\rangle \rightarrow |f(x)\rangle, \tag{7}$$

where the second register is treated as an internal part of the network (temporary register).

III. NETWORK ARCHITECTURE

Quantum networks for basic arithmetic operations can be constructed in a number of different ways. Although almost any nontrivial quantum gate operating on two or more qubits can be used as an elementary building block of the networks [9] we have decided to use the three gates described in Fig. 1, hereafter refered to as *elementary gates*. None of these gates is universal for quantum computation; however, they suffice to build any Boolean functions as the Toffoli gate alone suffices to support any *classical* reversible computation. The NOT and the control-NOT gates are added for convenience (they can be easily obtained from the Toffoli gates).

A. Plain adder

The addition of two registers $|a\rangle$ and $|b\rangle$ is probably the most basic operation, in the simplest form it can be written as

$$|a,b,0\rangle \rightarrow |a,b,a+b\rangle. \tag{8}$$

Here we will focus on a slightly more complicated (but more useful) operation that rewrites the result of the computation into the one of the input registers, which is the usual way additions are performed in conventional irreversible hardware; i.e.,

$$|a,b\rangle \rightarrow |a,a+b\rangle. \tag{9}$$

As one can reconstruct the input (a,b) out of the output $(a,a+b)$, there is no loss of information, and the calculation

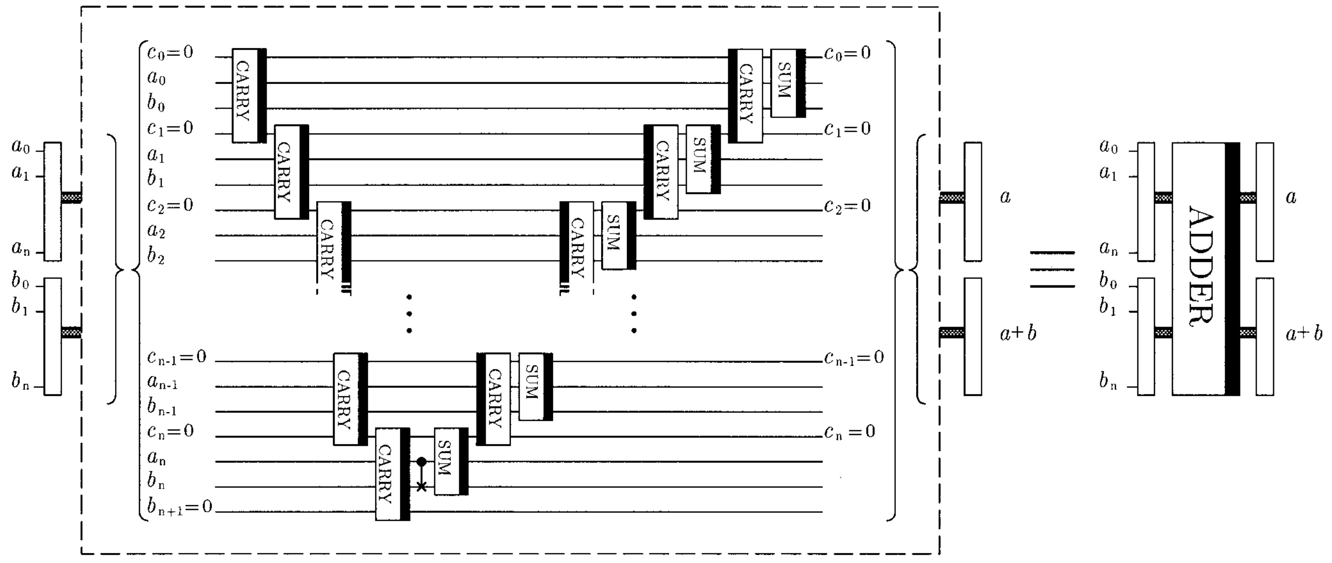


FIG. 2. Plain adder network. In the first step, all the carries are calculated until the last carry gives the most significant digit of the result. Then all these operations apart from the last one are undone in reverse order, and the sum of the digits is performed correspondingly. Note the position of a thick black bar on the right- or left-hand side of basic carry and sum networks. A network with a bar on the left side represents the reversed sequence of elementary gates embedded in the same network with the bar on the right side.

can be implemented reversibly. To prevent overflows, the second register (initially loaded in state $|b\rangle$) should be sufficiently large, i.e., if both a and b are encoded on n qubits, the second register should be of size $n+1$. In addition, the network described here also requires a temporary register of size $n-1$, initially in state $|0\rangle$, to which the carries of the addition are provisionally written (the last carry is the most significant bit of the result and is written in the last qubit of the second register).

The operation of the full addition network is illustrated in Fig. 2 and can be understood as follows.

We compute the most significant bit of the result $a+b$. This step requires computing all the carries c_i through the relation $c_i \leftarrow a_i$ AND b_i AND c_{i-1}, where a_i, b_i, and c_i represent the ith qubit of the first, second, and temporary (carry) register, respectively. Figure 3(a) illustrates the subnetwork that effects the carry calculation.

Subsequently we reverse all these operations (except for the last one which computed the leading bit of the result) in order to restore every qubit of the temporary register to its initial state $|0\rangle$. This enables us to reuse the same temporary register, should the problem, for example, require repeated additions. During the resetting process the other n qubits of the result are computed through the relation $b_i \leftarrow a_i$ XOR b_i XOR c_{i-1} and stored in the second register. This operation effectively computes the n first digits of the sum [the basic network that performs the summation of three qubits modulo 2 is depicted in Fig. 3(b)].

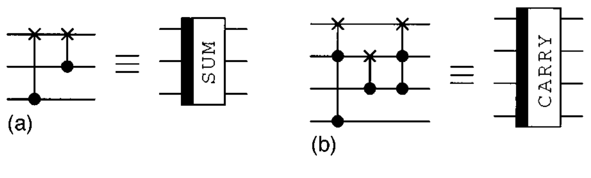


FIG. 3. Basic carry and sum operations for the plain addition network. (a) the carry operation (note that the carry operation perturbs the state of the qubit b). (b) the sum operation.

If we reverse the action of the above network (i.e., if we apply each gate of the network in the reversed order) with the input (a,b), the output will produce $(a,a-b)$ when $a \geqslant b$. When $a<b$, the output is $(a,2^{n+1}-(b-a))$, where $n+1$ is the size of the second register. In this case the most significant qubit of the second register will always contain 1. By checking this ''overflow bit'' it is therefore possible to compare the two numbers a and b; we will use this operation in the network for modular addition.

B. Adder modulo N

A slight complication occurs when one attempts to build a network that effects

$$|a,b\rangle \rightarrow |a,a+b \bmod N\rangle, \tag{10}$$

where $0 \leqslant a,b<N$. As in the case of the plain adder, there is no *a priori* violation of unitarity since the input (a,b) can be reconstructed from the output $(a,a+b \bmod N)$, when $0 \leqslant a,b<N$ (as will always be the case). Our approach is based on taking the output of the plain adder network, and subtracting N, depending on whether the value $a+b$ is bigger or smaller than N. The method, however, must also accomodate a superposition of states for which some values $a+b$ are bigger than N and some smaller than N.

Figure 4 illustrates the various steps needed to implement modular addition. The first adder performs a plain addition on the state $|a,b\rangle$ returning $|a,a+b\rangle$; the first register is then swapped with a temporary register formerly loaded with N, and a subtractor (i.e., an adder whose network is run backwards) is used to obtain the state $|N,a+b-N\rangle$. At this stage the most significant bit of the second register indicates whether or not an overflow occurred in the subtraction, i.e., whether $a+b$ is smaller than N or not. This information is

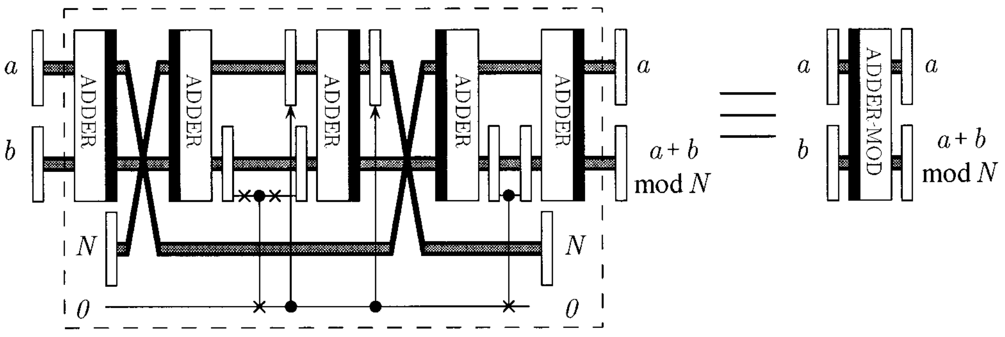


FIG. 4. Adder modulo N. The first and the second network add a and b together and then subtract N. The overflow is recorded into the temporary qubit $|t\rangle$. The next network calculates $(a+b)\text{mod}N$. At this stage we have extra information about the value of the overflow stored in $|t\rangle$. The last two blocks restore $|t\rangle$ to $|0\rangle$. The arrow before the third plain adder means that the first register is set to $|0\rangle$ if the value of the temporary qubit $|t\rangle$ is 1 and is otherwise left unchanged (this can be easily done with control-NOT gates, as we know that the first register is in the state $|N\rangle$). The arrow after the third plain adder resets the first register to its original value (here $|N\rangle$). The significance of the thick black bars is explained in the caption of Fig. 2.

''copied'' into a temporary qubit $|t\rangle$ (initially prepared in state $|0\rangle$) through the control-NOT gate. Conditionally on the value of this last qubit $|t\rangle$, N is added back to the second register, leaving it with the value $a+b\text{mod}N$. This is done by either leaving the first register with the value N (in case of overflow), or resetting it to 0 (if there is no overflow) and then using a plain adder. After this operation, the value of the first register can be reset to its original value and the first and the temporary register can be swapped back, leaving the first two registers in state $|a,a+b\text{mod}N\rangle$ and the temporary one in state $|0\rangle$. At this point the modular addition has been computed, but some information is left in the temporary qubit $|t\rangle$ that recorded the overflow of the subtraction. This temporary qubit cannot be reused in a subsequent modular addition, unless it is coherently reset to zero. The last two blocks of the network take care of this resetting: first the value in the first register $(=a)$ is subtracted from the value in the second $(=a+b\text{mod}N)$ yielding a total state $|a,(a+b\text{mod}N)-a\rangle$. As before, the most significant bit of the second register contains the information about the overflow in the subtraction, indicating whether or not the value N was subtracted after the third network. This bit is then used to reset the temporary bit $|t\rangle$ to $|0\rangle$ through a second control-NOT gate. Finally, the last subtraction is undone, returning the two registers to the state $|a,a+b\text{mod}N\rangle$.

C. Controlled-multiplier modulo *N*

Function $f_{a,N}(x)=ax\text{mod}N$ can be implemented by repeated conditional additions (modulo N): $ax=2^0ax_0+2^1ax_1+\cdots+2^{n-1}ax_{n-1}$. Starting from a register initially in the state $|0\rangle$, the network consists simply of n stages in which the value 2^ia is added conditionally, depending on the state of the qubit $|x_i\rangle$. Figure 5 shows the corresponding network; it is slightly complicated by the fact that we want the multiplication to be effected conditionally upon the value of some external qubit $|c\rangle$, namely, we want to implement

$$|c;x,0\rangle\to\begin{cases}|c;x,a\times x\text{mod}N\rangle & \text{if } |c\rangle=|1\rangle\\ |c;x,x\rangle & \text{if } |c\rangle=|0\rangle.\end{cases}\tag{11}$$

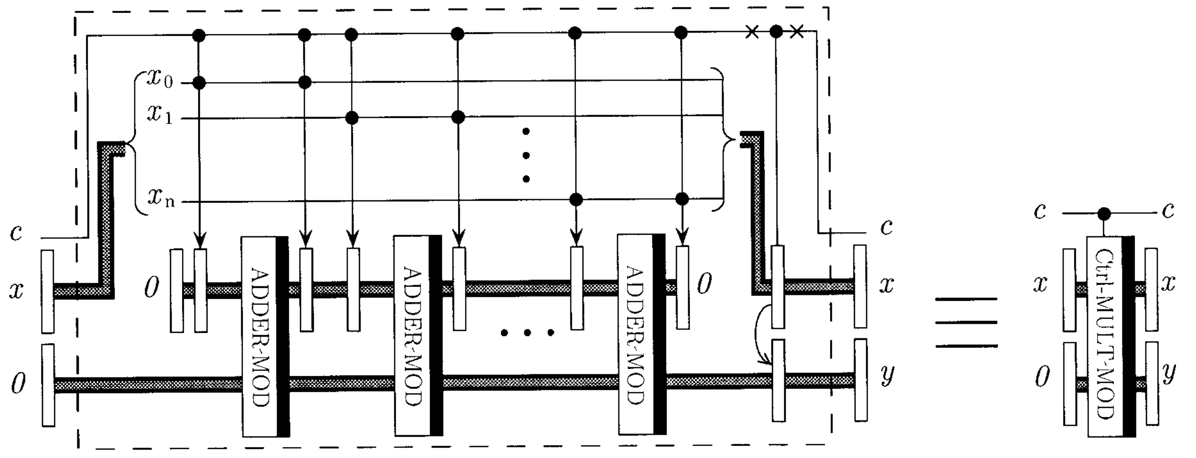


FIG. 5. Controlled multiplication modulo N consists of consecutive modular additions of 2^ia or 0 depending on the values of c and x_i. The operation before the ith modular adder consists in storing $2^{i-1}a$ or 0 in the temporary register depending on whether $|c,x_i\rangle=|1,1\rangle$ or not, respectively. Immediately after the addition has taken place, this operation is undone. At the end, we copy the content of the input register in the result register only if $|c\rangle=|0\rangle$, preparing to account for the fact that the final output state should be $|c;x,x\rangle$ and not $|c;x,0\rangle$ when $c=0$. The signification of the thick black bars is given in the caption of Fig. 2.

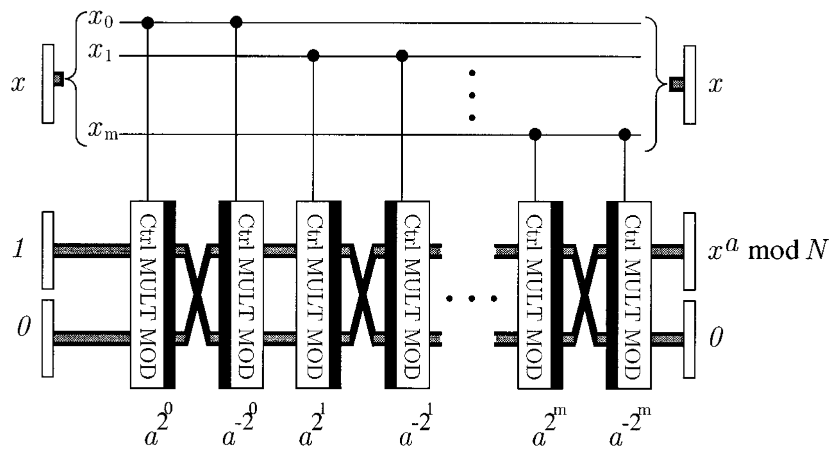


FIG. 6. Modular exponentiation consists of successive modular multiplications by a^{2^i}. The even networks perform the reverse control modular multiplication by inverse of $a^{2^i} \bmod N$ thus resetting one of the registers to zero and freeing it for the next control modular multiplication. The signification of the thick black bars is given in the caption of Fig. 2.

To account for this fact at the ith modular addition stage the first register is loaded with the value $2^i a$ if $|c,x_i\rangle = |1,1\rangle$ and with value 0 otherwise. This is done by applying the Toffoli gate to the control qubits $|c\rangle$ and $|x_i\rangle$ and the appropriate target qubit in the register; the gate is applied each time value "1" appears in the binary form of the number $2^i a$.

Resetting the register to its initial state is done by applying the same sequence of the Toffoli gates again (the order of the gates is irrelevant as they act on different target qubits). If $|c\rangle = |0\rangle$ only 0 values are added at each of the n stages to the result register, giving state $|c;x,0\rangle$. Since we want the state to be $|c;x,x\rangle$ we copy the content of the input register to the result register if $|c\rangle = |0\rangle$. This last operation is performed by the rightmost elements of the network of Fig. 5. The conditional copy is implemented using an array of Toffoli gates.

D. Exponentiation modulo N

A reversible network that computes the function $f_{a,N}(x) = a^x \bmod N$ can now be designed using the previous constructions. Notice first that a^x can be written as $a^x = a^{2^0 x_0} \cdot a^{2^1 x_1} \cdot \ \cdots \ \cdot a^{2^{m-1} x_{m-1}}$, thus modular exponentiation can be computed by setting initially the result register to $|1\rangle$, and successively effecting n multiplications by a^{2^i} (modulo N) depending on the value of the qubit $|x_i\rangle$; if $x_i = 1$, we want the operation

$$|a^{2^0 x_0 + \cdots + 2^{i-1} x_{i-1}}, 0\rangle \rightarrow |a^{2^0 x_0 + \cdots + 2^{i-1} x_{i-1}}, a^{2^0 x_0 + \cdots + 2^{i-1} x_{i-1}} \cdot a^{2^i}\rangle \tag{12}$$

to be performed, otherwise, when $x_i = 0$ we just require

$$|a^{2^0 x_0 + \cdots + 2^{i-1} x_{i-1}}, 0\rangle \rightarrow |a^{2^0 x_0 + \cdots + 2^{i-1} x_{i-1}}, a^{2^0 x_0 + \cdots + 2^{i-1} x_{i-1}}\rangle. \tag{13}$$

Note that in both cases the result can be written as $|a^{2^0 x_0 + \cdots + 2^{i-1} x_{i-1}}, a^{2^0 x_0 + \cdots + 2^i x_i}\rangle$. To avoid an accumulation of intermediate data in the memory of the quantum computer, particular care should be taken to erase the partial information generated. This is done, as explained in Sec. II, by running backwards a controlled multiplication network with the value $a^{-2^i} \bmod N$. This quantity can be efficiently precomputed in a classical way [8]. Figure 6 shows the network for a complete modular exponentiation. It is made out of m stages; each stage performs the following sequence of operations:

$$\begin{gathered} |a^{2^0 x_0 + \cdots + 2^{i-1} x_{i-1}}, 0\rangle \rightarrow \text{(multiplication)} \\ |a^{2^0 x_0 + \cdots + 2^{i-1} x_{i-1}}, a^{2^0 x_0 + \cdots + 2^i x_i}\rangle \rightarrow \text{(swapping)} \\ |a^{2^0 x_0 + \cdots + 2^i x_i}, a^{2^0 x_0 + \cdots + 2^{i-1} x_{i-1}}\rangle \rightarrow \text{(resetting)} \\ |a^{2^0 x_0 + \cdots + 2^i x_i}, 0\rangle. \end{gathered} \tag{14}$$

IV. NETWORK COMPLEXITY

The size of the described networks depends on the size of their input n. The number of elementary gates in the plain adder, the modular addition, and the controlled modular addition network scales linearly with n. The controlled modular multiplication contains n controlled modular additions, and thus requires of the order of n^2 elementary operations. Similarly the network for exponentiation contains of the order of n controlled modular multiplications and the total number of elementary operations is of the order of n^3. The multiplicative overhead factor in front depends very much on what is considered to be an elementary gate. For example, if we choose the control-NOT to be our basic unit then the Toffoli gate can be simulated by 6 control-NOT gates [10].

Let us have a closer look at the memory requirements for the modular exponentiation; this can help to assess the difficulty of quantum factorization. We set n to be the number of bits needed to encode the parameter N of Eq. (1). In Shor's algorithm, x can be as big as N^2, and therefore the register needed to encode it requires up to $2n$ qubits. Not counting the two input registers and an additional bit to store the most significant digit of the result, the plain adder network requires an extra $(n-1)$-qubit temporary register for storing temporary (carry) qubits. This register is reset to its initial value, $|0\rangle$, after each operation of the network and can be reused later. The modular addition network, in addition to the temporary qubit needed to store overflows in subtractions, requires another n-qubit temporary register; in total this makes two n-qubit temporary registers for modular addition. Controlled modular multiplication is done by repeated modular additions, and requires three temporary n-qubit registers: one for its own operation and two for the modular addition (controlled modular multiplication also requires a temporary qubit used by the modular addition network). Finally, the network for exponentiation needs four temporary n–qubit registers, one for its own operation and three for the controlled modular multiplication (plus an additional qubit used by the modular addition). Altogether the total number of qubits required to perform the first part of the factorization algorithm is $7n+1$, where $2n$ qubits are used to store x, n qubits store the result $a^x \mathrm{mod} N$, and $4n+1$ qubits are used as temporary qubits.

The networks presented in this paper are by no means the only or the most optimal ones. The notion of optimal network strongly depends on its experimental realization and of the type of algorithm it implements. If decoherence can be kept at low levels and if storing quantum information is easy within a given experimental framework, then one may choose to use many temporary qubits and minimize the number of gate operations. On the other hand, if space is an expensive resource, then it is desirable to minimize the number of temporary qubits at the expense of longer time of computation. In this work we focused on minimizing the number of required qubits showing that for the modular exponentiation $a^x \mathrm{mod} N$ there exist networks for which this number grows linearly with the size of N. There are many ways to construct operation such as $a^x \mathrm{mod} N$, given parameters a and N. Usually a dedicated network composed of several subunits does not have to be a simple sum of the subunits. In the modular exponentiation, for example, it is relatively easy to reduce the memory, i.e., the constant overhead factor (7 in our case) by noting that the first register in the plain adder network always stores specific classical values: either 0 or N. The same holds for the temporary register in the adder modulo N which always stores either 0 or $2^i a \mathrm{mod} N$. There is no need to use a full quantum register for this: a classical register plus a single qubit (that keeps track of the entanglement) are sufficient. This reduces the number of qubits to $5n+2$. One further register can be removed by using the addition network that does not require a temporary register [11]; the trick is to use the n-bit Toffoli gates to add n-bit numbers. If the difficulty of the practical implementations of the n-bit Toffoli gates is comparable to that of the regular Toffoli gate, then this can be a good way of saving memory. All together the number of qubits can be reduced from $7n+1$ to $4n+3$. This means that apart from the register storing x and another one storing $a^x \mathrm{mod} N$ we need additional $n+3$ temporary qubits to perform quantum modular exponentiation in Shor's algorithm. The required memory grows only as a linear function of the size of N.

In some cases, gain in space can be obtained by making use of so-called divide-and-conquer techniques [12] in which simple tasks are split in subtasks for which a smaller number of temporary qubits is needed. For instance, in the case of the simple n bits adder one can save temporary qubit by first effecting the addition on the low halves of the input ($n/2$ least significant bits). This requires approximately $n/2$ temporary qubits. These qubits, after being erased by the usual technique of reversing the computation, could be used to add the upper halves of the input and then to combine the two results. This method can be iterated and applied in turn on each of the subadditions. Let us mention, however, that in the networks presented in this paper, this technique can be applied only to the simple adder. Also note that if we are willing to implement n-bit Toffoli gates in our network we can avoid the problem of the temporary register altogether (see discussion above [11]).

The multiplication algorithm presented in the paper is a simple algorithm which is efficient for multiplying small numbers. For large numbers there exist more efficient classical algorithms (e.g., Schönhage-Strassen [13]). However, a reversible implementation of them is far beyond the scope of this work and is justified only for multiplying numbers of size of the order of 500 bits or more [12].

V. CONCLUSION

In this paper we have explicitly constructed quantum networks performing elementary arithmetic operations including the modular exponentiation which dominates the overall time and memory complexity in Shor's quantum factorization algorithm. Our network for the modular exponentiation achieves only a linear growth of auxiliary memory by exploiting the fact that $f_{a,N}(x)=ax \mathrm{mod} N$ is a bijection (when a and N are coprime) and can be made reversible by simple auxiliary computations. In more practical terms our results indicate that with the ''trapped ions computer'' [14] about 20 ions suffice (at least in principle) to factor $N=15$. Needless to say, the form of the actual network that will be used in the first quantum computer will greatly depend on the type of technology employed; the notion of an optimal network is architecture dependent and any further optimization has to await future experimental progress.

Note added in proof. After submission of this paper, we learned of similar work by D. Beckman, A. N. Chan, S. Devabhaktoni, and J. Preskill.

ACKNOWLEDGMENTS

V.V. thanks the Royal Society for financial support which enabled him to undertake the research project on the subject of the paper. A.B. acknowledges the financial support of the Berrow Fund at Lincoln College, Oxford. The authors would like to thank D. Deutsch, D. DiVincenzo, S. Gardiner, H.J. Kimble, P.L. Knight, E. Knill, T. Pelizzari, and P. Zoller for useful discussions.

[1] D. Deutsch, Proc. R. Soc. London A **400**, 97 (1985).
[2] D. Deutsch and R. Jozsa, Proc. R. Soc. London A **439**, 553 (1992); E. Bernstein and U. Vazirani (unpublished); D. S. Simon, in *Proceedings of the 35th Annual Symposium on the Foundations of Computer Science*, edited by S. Goldwasser (IEEE Computer Society Press, Los Alamitos, CA, 1994), p. 16.
[3] P. W. Shor, in *Proceedings of the 35th Annual Symposium on the Theory of Computer Science* (Ref. [2]), p. 124.
[4] D. Deutsch, *The Fabric of Reality* (Viking-Penguin Publishers, London, in press).
[5] D. Deutsch, Proc. R. Soc. London A **425**, 73 (1989).

[6] R. Landauer, IBM J. Res. Dev. **5**, 183 (1961); C. H Bennett, *ibid.* **32**, 16 (1988); T. Toffoli, Math. Syst. Theory **14**, 13 (1981).

[7] C. H. Bennett, SIAM J. Comput. **18** (4), 766 (1989); R. Y. Levine and A. T. Sherman, *ibid.* **19** (4), 673 (1990).

[8] D. E. Knuth, *The Art of Computer Programming, Volume 2: Seminumerical Algorithms* (Addison-Wesley, New York, 1981).

[9] A. Barenco, Proc. R. Soc. London A **449**, 679 (1995); T. Sleator and H. Weinfurter, Phys. Rev. Lett. **74**, 4087 (1995); D. Deutsch, A. Barenco, and A. Ekert, Proc. R. Soc. London A **449**, 669 (1995); S. Lloyd, Phys. Rev. Lett. **75**, 346 (1995).

[10] A. Barenco, C.H. Bennett, R. Cleve, D. P. DiVicenzo, N. Margolus, P. Shor, T. Sleator, J. Smolin, and H. Weinfurter, Phys. Rev. A **52**, 3457 (1995).

[11] S. A. Gardiner, T. Pelizzari, and P. Zoller (private communication).

[12] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *Data Structures and Algorithms* (Addison-Wesley, New York, 1983).

[13] A. Schoenhage and V. Strassen, Computing **7**, 281 (1971).

[14] J. I. Cirac and P. Zoller, Phys. Rev. Lett. **74**, 4091 (1995).