Available online at www.sciencedirect.com
ScienceDirect
Information and Computation 205 (2007) 967–981
Information and Computation
www.elsevier.com/locate/ic

Efficient and exact quantum compression

John H. Reif [*,1], Sukhendu Chakraborty [2]

Department of Computer Science, Duke University, Durham, NC 27708-0129, USA

Received 2 June 2005; revised 11 July 2006
Available online 2 February 2007

Abstract

We present a divide and conquer based algorithm for optimal quantum compression/decompression, using $O(n(\log^4 n)\log\log n)$ elementary quantum operations. Our result provides the first quasi-linear time algorithm for asymptotically optimal (in size and fidelity) quantum compression and decompression. We also outline the quantum gate array model to bring about this compression in a quantum computer. Our method uses various classical algorithmic tools to significantly improve the bound from the previous best known bound of $O(n^3)$ for this operation.

Keywords: Quantum computing; Quantum compression; Algorithm

1. Introduction

1.1. Quantum computation

Quantum computation (QC) is a computing model that applies quantum mechanics to do computation. (Computations and methods not making use of quantum mechanics will be termed *classical*). A single molecule (or collection of particles and/or atoms) may have n degrees of freedom known as *qubits*. Associated with each fixed setting X of the n qubits to boolean values is a *basis state* denoted $|a\rangle$. Quantum mechanics allows for a linear superposition of these basis states to exist simultaneously. Each basis state $|a\rangle$ of the superposition is assigned a given complex amplitude α; this is denoted $\alpha|a\rangle$. *Unitary transformations* are reversible operations on the superpositions which can be represented by unitary matrices A (e.g., permutation matrices, rotation matrices, and the matrices of Fourier transforms) where $AA^* = A^*A = I$ (we use A^* to denote the conjugate transpose (*hermitian*) of matrix A). The sum of the squares of the magnitudes of the amplitudes of all basis states is 1. This sum remains invariant due to the application of unitary transformations. The Hilbert space H_n is the set of all possible such linear superpositions.

* Corresponding author. Fax: +1 919 660 6519.
E-mail addresses: reif@cs.duke.edu (J.H. Reif), schak@cs.duke.edu (S. Chakraborty).
[1] The work is supported by NSF EMT Grants CCF-0523555 and CCF-0432038.
[2] Present address: Oracle Corp., 400 Oracle Pkwy., Redwood Shores, CA 94065, USA.

0890-5401/$ - see front matter © 2007 Elsevier Inc. All rights reserved.
doi:10.1016/j.ic.2007.01.005

Operations executed on these superpositions allowed by QC can be classified into two main categories: (i) *unitary operations*, and (ii) *observation operations*, which allow for the (strong) measurement of each qubit, providing a mapping from the current superposition to a superposition where the measured qubit is assigned a boolean value with probability given by the square of the amplitude of the qubit in its original superposition. *Elementary unitary operations* that suffice for any quantum computation over qubits [2,12] include a conditional form of the conditional XOR operation $\oplus$, the boolean operation NOT, and a constant boolean operation yielding 0. The *time bound* for a quantum computations is defined to be the number of such elementary unitary operations.

Deutsch [11] defined a quantum computing model known as a *quantum gate array* which allows execution of a sequence of quantum gates, where each input is a qubit, and each gate computes a unitary transformation. We will assume this model, with the above elementary unitary operations for the gates.

1.2. Classical lossless compression

Suppose n characters from a finite alphabet Σ are each sampled independently over some probability distribution p. In classical information theory, the Shannon entropy of each character is $H_\Sigma(p) = -\sum_{a\in\Sigma} p(a)\log p(a)$. The *compression rate* is the ratio between the length of an uncompressed string and the length of the compressed (binary) string.

There are many algorithms for universal data compression in classical domain. One such efficient and widely used data compression algorithm is due to Ziv and Lempel [9].

They present a simple linear time lossless compression algorithm having an asymptotic compression rate approaching the source's entropy; that is allows a string of length n to be losslessly compressed to a bit string of length asymptotic approaching $H_\Sigma(p)n$ for large n. In the first pass, they use a parsing scheme to encode the source string into unique prefixes. In the second pass, they use this encoded information to recover the original string without error.

Unfortunately, developing a quantum analog of Ziv and Lempel's universal data compression in the classical domain is not trivial since keeping track of prefixes require multiple measurements of the same qubits which makes the operation irreversible. The same difficulties appear to hold for all other known classical compression algorithms [9].

1.3. Quantum lossless compression

It may be very advantageous to decrease, where possible by compression methods, the number of qubits used for quantum communication and storage. Holevo [14], Fuchs and Caves [13] and Reif [21] have results that imply that quantum methods can not increase the bandwidth for transmission of classical information. However, entangled quantum states can be compressed much more than possible via classical lossless compression. Following Schumacher [22], we assume there is a finite quantum state ensemble (Σ', p) which is a *mixed state* consisting of a finite number of qubit states $\Sigma' = \{|a_0\rangle, \ldots, |a_{|\Sigma'|-1}\rangle\}$, where each $|a_i\rangle \in \Sigma'$ has probability p_i. The compressor is assumed to act on blocks of n qubits (so is a block compressor), and is assumed to know this underlying ensemble (Σ', p). The *density matrix* of (Σ', p) is an $|\Sigma'| \times |\Sigma'|$ matrix $\rho = \sum_{i=0}^{|\Sigma'|-1} p_i |a_i\rangle\langle a_i|$, where $|a_i\rangle\langle a_i|$ is the projection operator on the signal state $|a_i\rangle$. The *von Neumann entropy* [20,22] corresponding to (Σ', p) is $H_{VN}(\rho) = -Tr(\rho\log\rho)$ where Tr is the trace operator. In general, the Shannon entropy $H_{\Sigma'}(p)$ is greater than or equal to the von Neumann entropy. These entropies are equal only when the states in Σ' are mutually orthogonal.

The unitary compression and decompression mappings need to preserve the number of bits (some of which are ignored). An *n-to-n' quantum compressor* is a unitary transformation that maps n-qubit strings to n-qubit strings; the first n' qubits that are output by the compressor are taken as the compressed version of its input, and the remaining n–n' qubits are discarded. An *n'-to-n decompressor* is a unitary transformation that maps n-qubit strings to n-qubit strings; the first n' qubits input to the decompressor are the compressed version of the uncompressed n qubits, and the remaining n–n' qubits are all 0. The *source* to the compression scheme is assumed to be a sequence of n qubits sampled independently from (Σ', p). The *observed output* is the result of first compressing the input qubits, then decompressing them, and finally measurement of the result (over a basis

containing the n inputs).[1] The *fidelity* of the compression scheme is the probability that the observed output is equal to the original input (that is the probability that the original qubits are correctly recovered, from the compressed qubits). Our goal is a quantum compression with both high fidelity and a high compression rate.

Schumacher [22] gave a *quantum coding theorem* which provided asymptotically optimal (in size and fidelity) compression of a sequence of qubits independently sampled from a finite quantum state ensemble (Σ', p).

Theorem 1.1. *The quantum coding theorem* [22], *states that for any* $\epsilon, \delta > 0$ *and sufficiently large* n, (*i*) *there is a quantum compression scheme that achieves asymptotic compression rate* $\leq (H_{VN}(\rho) + \delta)$ *with fidelity at least* $1 - \epsilon$ *and*, (*ii*) *any quantum compression scheme that gives asymptotic compression rate* $\leq (H_{VN}(\rho) - \delta)$ *has fidelity* $< \epsilon$.

In other words, in the limit of large code-block size, the source's von Neumann entropy $H_{VN}(\rho)$ is asymptotically the number of qubits per source state which is necessary and sufficient to encode the output of the source with arbitrarily high fidelity. Given a known finite quantum state ensemble (Σ', p), Schumacher's compression scheme assumes a known basis for which the density matrix ρ is diagonal, with non-increasing values along the diagonal. The proof of the Schumacher quantum coding theorem and its refinements by Jozsa and Schumacher [15], Barnum et al. [1], and Szeto [25] make use of the existence of a *typical subspace* Λ within a Hilbert space of n qubits over a source of von Neumann entropy $H_{VN}(\rho)$. These proofs are not constructive.

Bennett [6] gave a constructive method for doing Schumacher compression. He observed that the Schumacher compression can be done by a unitary mapping to a basis for which the density matrix ρ is diagonal (in certain simple cases the density matrix ρ is already diagonal, e.g., when the input is a set of n identical qubits) followed by certain combinatorial computation which we will call the *Schumacher compression function.* The Schumacher compression function S simply orders the basis states first by the number of ones (from smallest to largest) that are in the binary expansion of the bits and then refines this order by a lexical sort of the binary expansion of the bits. That is, all strings with i ones are mapped before all strings with $i + 1$ ones, and those strings with the same number of ones are lexically ordered. Note that for any given value X of the qubits, this transformation $S(X)$ is simply a deterministic mapping from an n bit sequence to a n' bit sequence defined by a combinatorial computation. In particular, given an n bit binary string X, the Schumacher compression function $S(X)$ is the number of n bit strings so ordered before X. It is easy to show that the Schumacher compression function is a permutation. Since it is a permutation, it is a bijective function which is uniquely reversible, and also is a unitary transformation.

To ensure that the overall transformation (for all the states) is a quantum computation, it is essential that the Schumacher compression function be done using only reversible, quantum-coherent elementary operations. Bennett et al. [7] gave a polynomial time quantum algorithm for the related problem of extraction of only classical information from a quantum noiseless coding. Cleve and DiVincenzo [10] then developed the first polynomial time algorithm for Schumacher compression of n qubits. In particular, they explicitly computed the bijective function defined by the Schumacher compression function and its reverse using $O(n^3)$ reversible, elementary unitary operations.

The Schumacher quantum coding theorem assumes the compressor knows the source. Jozsa et al. [16] recently gave a generalization of the Schumacher compression to the case where the compressor does not know the source, thus providing the first asymptotically optimal universal algorithm for quantum compression. Also, Braustein et al. [8] have recently given a fast algorithm for a quantum analog of Huffman coding, but do not provide a proof that this coding gives asymptotically optimal quantum compression (that is, reaches the von Neumann entropy), as provided by Schumacher compression.

1.4. Organization and results

In Section 2, we give an efficient deterministic algorithm for a modified Schumacher compression encoding function $S'(X)$ (also with asymptotically optimal size and fidelity) using $O(n(\log^4 n)\log\log n)$ boolean operations. Next, in Section 3 we show that, exploiting the inherent binary tree structure of our modified quantum

[1] One of the most interesting parts of quantum compression, is that ρ could represent the reduced state of some larger system, and the compression will preserve the entanglement within this larger system. If one just measures after compression, there may as well have been a measurement before compression, which allows a completely classical scheme.

compression algorithm and using known efficient quantum algorithms for conditional boolean operations and integer arithmetic, we can execute our quantum compression algorithm on a quantum gate array in asymptotically the same number ($O(n(\log^4 n)\log\log n)$) of operations as required by our reversible algorithm for our modified Schumacher encoding function. Then, in Section 4, we show how the various subroutines required by our algorithm can be made reversible, and the modified Schumacher encoding and decoding can be efficiently computed by a quantum computer within the $O(n(\log^4 n)\log\log n)$ elementary unitary steps. In the same section we give recursive, reversible algorithms for some required combinatorial and double combinatorial sums. This result is a considerable reduction from the previous best time bounds of $O(n^3)$ for Schumacher compression due to Cleve and DiVincenzo [10]. Due to the use of Schumacher-type encoding, our compression and fidelity bounds are asymptotically optimal. For simplicity, our algorithm assumes the compressor knows the source, but can be extended to a asymptotically optimal universal algorithm for quantum compression where the compressor does not know the source, using the techniques of Jozsa et al. [16].

2. Deterministic computation of a modified Schumacher compression

The number of n bit (henceforth, we use *qubit* and *bit* interchangeably when the context is clear) numbers with exactly m ones, for $1 \le m \le n$, is $\binom{n}{m} = \frac{n!}{(n-m)!m!}$. For $m < 0$ or $m > n$, we define $\binom{n}{m} = 0$. For any $1 \le m \le n$, the number of n bit numbers with $< m$ ones is $1 + \sigma_{n,m}$ where $\sigma_{n,m} = \sum_{i=1}^{m-1} \binom{n}{i}$. In Section 4, we give an efficient reversible algorithm for the combinatorial sum $\sigma_{n,m}$, using $O(M(m\log n)\log m)$ boolean operations, where $M(N) = O(N\log N\log\log N)$.

Let X and Y be n bit strings, and suppose X has exactly m ones. Let $X = X'X''$, and $Y = Y'Y''$ where X', Y' each have $n' = \lfloor n/2 \rfloor$ bits and X'', Y'' each have $n'' = n - n' = \lceil n/2 \rceil$ bits. Let $Y < X$ (we say Y *is lexically less than* X) if either (a) Y has less than m ones, or (b) Y has m ones and either (i) $Y' < X'$, or (ii) $Y' = X'$ but $Y'' < X''$.

For n bit X with m ones, we will redefine our compression function $S'(X) = S'(X, n, m)$ to be the number of n bit binary numbers Y such that $Y < X$. Since our modified Schumacher encoding $S'(X)$ simply is a permutation of Schumacher's original encoding $S(X)$, this modified Schumacher encoding $S'(X)$ clearly has the same fidelity as the original Schumacher compression.

Let $L(X) = L(X, n, m)$ be the number of distinct n bit binary numbers, with exactly m ones, that are lexically less than X. (Note that, $S'(X, n, m)$ and $L(X, n, m)$ each need only to be a function of X, and the additional arguments n and m will be used for notational convenience in the proofs.) For $n = 1$, $S'(0) = 0$ and $S'(1) = 1$ and so in this case $S'(X) = X$ is the identity map. Also, clearly if X has any number n of bits but $m = 0$ ones, then $S'(X) = X$ again is the identity map. For the general case $n > 1$ and $m \ge 1$, since every n bit binary number with less than m ones are lexically less than X, and there are $1 + \sigma_{n,m}$ such numbers, it follows that $S'(X, n, m) = 1 + \sigma_{n,m} + L(X, n, m)$.

The following Lemma gives the divide and conquer approach to compute $L(X, n, m)$.

Lemma 2.1. *Let X be any n bit binary number with m ones. If $m = 0$ or $n = 1$ then $L(X, n, m) = 0$. Otherwise, if $m \ge 1$ and $n > 1$ then $X = X'X'$,' where X' has $n' = \lfloor n/2 \rfloor$ bits and m' ones and X'' has $n'' = \lceil n/2 \rceil$ bits and $m'' = m - m'$ ones, then $L(X, n, m) = \binom{n''}{m} + \sigma'_{n,m,m'} + L(X', n', m')\binom{n''}{m''} + L(X'', n'', m'')$, where $\sigma'_{n,m,m'} = \sum_{i=1}^{m'-1} \binom{n'}{i}\binom{n''}{m-i}$.*

Proof by induction: Case $m = 0$: In the case $m = 0$ where X has no ones, so is all zeros, no other number of the same length is lexically less than X, so $L(X, n, m) = 0$.

Case $n = 1$: The case $n = 1$ is also trivial, since the number of 1 bit numbers lexically less than the number itself, having the same number of ones is always 0 irrespective of whether the number is 1 or 0.

Case $m \ge 1$ and $n > 1$: Let X be an n bit number with m ones. We can divide X as $X = X'X''$ where X' is a $n' = \lceil n/2 \rceil$ number with the first n' bits of X and X'' has the last $n'' = \lfloor n/2 \rfloor$ bits of X. Let the number of ones in X' be m' and that in X'' be $m'' = m - m'$. By the induction assumption (a) the number of n' bit numbers having

exactly m' ones that are lexically less than X' is given by $L(X', n', m')$, and (b) the number of n'' bit numbers having exactly m'' ones that are lexically less than X'' is given by $L(X'', n'', m'')$.

Our goal now is to determine the number of n bit numbers Y that are lexically less than X but have the same number m of ones as X. We can similarly divide Y as $Y = Y'Y''$ where Y' is a n' number with the first n' bits of Y and Y'' has the last n'' bits of Y. In this case $Y < X$ if either:

(i) $Y' < X'$, or
(ii) $Y' = X'$ but $Y'' < X''$.

The number of n bit numbers satisfying case (i) can be divided into following three categories.

- The number of n bit numbers having no ones in the first n' bits , so Y' has no ones. This quantity is equal to $\binom{n''}{m}$ which is the number of arrangements of m ones in the last n'' bits .
- The number of n bit numbers that are less than m' ones (but at least one) in their first n' bits. This quantity is given by $\sigma'_{n,m,m'}$, where $\sigma'_{n,m,m'} = \sum_{i=1}^{m'-1} \binom{n'}{i}\binom{n''}{m-i}$ is the number of distinct n bit binary numbers with i ones in the first n' bits, where $1 \le i \le m' - 1$, and $m - i$ ones in the last n'' bits.
- The number of n' bit numbers that have m' ones in their first n' bits but are lexically less than X'. This quantity is given by the induction assumption $(= L(X', n', m'))$. But for each such arrangement we can have $\binom{n''}{m''}$ arrangements of the last n'' bits of X and all those numbers will be lexically less than X'. So the number of n bit numbers having exactly m' ones in the first n' bits but lexically less than X' is: $L(X', n', m')\binom{n''}{m''}$.

Hence, we conclude that the total number of n bit numbers where $Y' < X'$ (as considered in case (i)) is given by $\binom{n''}{m} + \sigma'_{n,m,m'} + L(X', n', m')\binom{n''}{m''}$.

The case (ii) is when $Y' = X'$ but $Y'' < X''$. In other words, we have to count the number of n bit numbers whose first n' bits are same$(= X')$ such that it has exactly m'' ones in the last n'' bits but are lexically less than X''. This is again given by the induction assumption $(= L(X'', n'', m''))$.

Thus we have considered all possible cases and therefore the total number of n bit numbers having exactly m ones which are lexically less than X is given by: $L(X, n, m) = \binom{n''}{m} + \sigma'_{n,m,m'} + L(X', n', m')\binom{n''}{m''} + L(X'', n'', m'')$. Hence, the Lemma is true for all n and $m \le n$. □

Since the sum $\sigma'_{n,m,m'}$ is a product of $O(m)$ numbers each with $O(\log n)$ bits, it follows that $\sigma'_{n,m,m'}$ is an $O(m \log n)$ bit number. In Section 4, we give an efficient reversible algorithm for the double combinatorial sum $\sigma'_{n,m,m'}$, using $O(M(m \log n) \log m)$ boolean operations (using a recursive method similar to our algorithm for evaluation of $\sigma_{n,m}$ also given in Section 4).

The above recurrences for $L(X, n, m)$ and $S'(X, n, m) = 1 + \sigma_{n,m} + L(X, n, m)$ can thus be bounded as $T(n, m) \le O(M(m \log n) \log m) + 2T(n/2, m)$. Here $M(N)$, is the time taken to multiply two N bit numbers. Hence we have:

Corollary 2.1. *The forward computation of $S'(X, n, m)$ uses $O(M(n \log n) \log m \log n) \le O(n(\log^4 n) \log \log n)$ boolean operations, where $M(N) \le O(N(\log N) \log \log N)$.*

Next, we observe that we can reversibly compute the inverse function of $S'(X)$ by the following Lemma.

Lemma 2.2. *The backward computation of $S'(X, n, m)$ costs asymptotically the same number of boolean operations as the forward computation $(= O(n(\log^4 n) \log \log n))$.*

Proof. Given $S' = S'(X)$, we can count the number of bits n of the input. We can also determine the number m of ones of X and $L(X, n, m)$ from S' simultaneously, by applying a trick shown in Section 3. The recursive formula and the inverse computation of $\sigma'_{n,m,m'}$ uses $O(M(m \log n) \log m)$ reversible boolean operations, as described in Section 4. Then we recursively determine the position of ones within X by recursively determining the position of ones within X' and X'', where X' has $n' = \lfloor n/2 \rfloor$ bits and m' ones and X'' has $n'' = \lceil n/2 \rceil$ bits. We do this for $m \ge 1$

and $n > 1$ by applying the recursive formula: $L(X,n,m) = \binom{n''}{m} + \sigma'_{n,m,m'} + L(X',n',m')\binom{n''}{m''} + L(X'',n'',m'')$. Since there are again $O(\log n)$ stages, each of which costs $O(M(m\log n)\log m) \leq O(n(\log^3 n)\log\log n)$, the time cost of this inverse computation is $\leq O(n(\log^4 n)\log\log n)$, which is asymptotically the same as the forward computation of the $S'(X,n,m)$ function. □

Hence, from the Corollary 2.1 and Lemma 2.2, we have the following important result.

Corollary 2.2. *We can reversibly compute in time* $O(n(\log^4 n)\log\log n)$ *the bijection Schumacher compression function* $S'(X)$.

Next in Section 3 we show that, we can execute our quantum compression algorithm on a quantum gate array in asymptotically the same number $O(n(\log^4 n)\log\log n)$ of elementary unitary operations as required by Corollary 2.2.

3. Quantum computation of the modified Schumacher compression using quantum gates

3.1. Forward computation of $S'(X)$

Given that we can reversibly compute in time $O(n(\log^4 n)\log\log n)$ the bijection $S'(X)$, we now show that we can compute $S'(X)$ in $O(n(\log^4 n)\log\log n)$ elementary unitary operations like the conditional form of the conditional XOR operation $\oplus$, the boolean operation NOT, and a constant boolean operation yielding 0 on a quantum gate array. Recall that for $n = 1$ or $m = 0$, then $S'(X) = X$. For simplicity, we also assume n is a power of 2. (If this later restriction is removed then we can still construct a binary tree generalization (as discussed later) except that now we would get a complete binary tree instead of a full binary tree.)

The pseudocode for the recursive algorithm to compute $S'(X)$ outlined in Section 2 is shown below. For our implementation we assume that we have three quantum registers to start with:
X : an n-bit register used as the input
L : an n-bit register (initialized to 0) used as the output
W : an $\lceil(logn)\rceil$-bit register (initialized to 0) used as the work register to store intermediate results

Our first goal is to compute the following operation on the registers (X,L,W): $(X,0,0) \rightarrow (X,S'(X),0)$. We do this by recursively computing $L(X,n,m)$ and assigning this value to register L, and from this we determine $S'(X)$ and update the register L to this value. In Section 4, we will see that we can achieve the desired final output state $(S'(X),0,0)$ with no additional storage for the input.

We assume a simple subroutine NUMONES that counts the number of ones in it's argument. Hence, if the size of the argument passed to NUMONES is of size n then the size of the output returned by it is $O(\log n)$.

In the recursive algorithm to compute $L(X,n,m)$, m, m' and m'' are work registers used for temporary storage and addition operations at each iteration of the while loop in Algorithm 1. The addition $(+)$ and the multiplication operations $(\cdot)$ are the standard arithmetic addition and multiplication of qubits as shown in Vedral et al. [27]. The notation $\leftarrow$ has been used specifically to differentiate quantum assignments from normal assignments denoted by $=$. Also the notation $X_{[i,j]}$ (respectively $L_{[i,j]}$) represents the substring of the register X (respectively L) starting from the ith bit to the jth bit. Note that since the register L is initialized to 0, so is $L_{[i,j]}$. Also, note that in the case $m = 0$ or $n = 1$, no bits of L are assigned to, so it remains 0.

The assignment:

$$L_{[k,k+2^i-1]} \leftarrow \binom{2^{i-1}}{m} + \sigma'_{2^i,m,m'} + L_{[k,k+2^{i-1}-1]} \cdot \binom{2^{i-1}}{m''} + L_{[k+2^{i-1},k+2^i-1]}$$

is justified by Lemma 2.1, and allows us an inductive proof of the recursive algorithm for computing $L(X,n,m)$. At each step we have $L_{[i,j]}(= L(X_{[i,j]},n_1,m_1))$, where n_1 and m_1 are the number of bits and number of ones in the $[i,j]$ substring of the register X. As we go higher in the recursion we use the value calculated in the earlier recursion step with no additional storage.

The next goal is to translate Algorithm 1 into a sequence of elementary quantum mechanical operations (again we assume that n is a power of 2).

Algorithm 1 Recursive algorithm to compute $L(X, n, m)$

for $i = 1$ to $\lceil \log n \rceil$ **do**
  $k = 1$
  while $k + 2^i - 1 \leq n$ **do**
    $m' \leftarrow \text{NUMONES}(X_{[k,k+2^{i-1}-1]})$
    $m'' \leftarrow \text{NUMONES}(X_{[k+2^{i-1},k+2^i-1]})$
    $m \leftarrow m' + m''$
    if $m > 0$ and $n > 1$ **then**
      Apply this recursive algorithm to compute $L(X_{[k,k+2^{i-1}-1]}, 2^{i-1}, m')$ with the side effect of assigning the result to $L_{[k,k+2^i-1]}$.
      Apply this recursive algorithm to compute $L(X_{[k+2^{i-1},k+2^i-1]}, 2^{i-1}, m'')$ with the side effect of assigning the result to $L_{[k+2^{i-1},k+2^i-1]}$.
      $L_{[k,k+2^i-1]} \leftarrow \binom{2^{i-1}}{m} + \sigma'_{2^i,m,m'} + L_{[k,k+2^{i-1}-1]} \cdot \binom{2^{i-1}}{m''} + L_{[k+2^{i-1},k+2^i-1]}$
      $k \leftarrow k + 2^i$
  end while
end for

The main constraint of these operations is that they must obey the reversibility criteria. Its easy to see that the algorithm being recursive has a built in binary tree structure. The calculation of the function $L(X, n, m)$ at each recursive step requires the values calculated at the previous step involving half the number of bits.

The overall organization of the logic gates for performing the tree based operation is shown in Fig. 1. The recursion in the equation of Lemma 2.1 is only due to the computation of the number of n bit numbers which have equal number of ones but are lexically less than X (i.e., $L(X, n, m)$). So, the tree performs this computation of $L(X, n, m)$ in a bottom up manner. At each level i of the tree, the register L stores the $L(X, 2^i, m_i)$ values in $n/2^i$ tuples of the bits of L. Here, m_i is the number of ones in the $n/2^i$ tuples of X. At the $(i + 1)$th level of the recursive

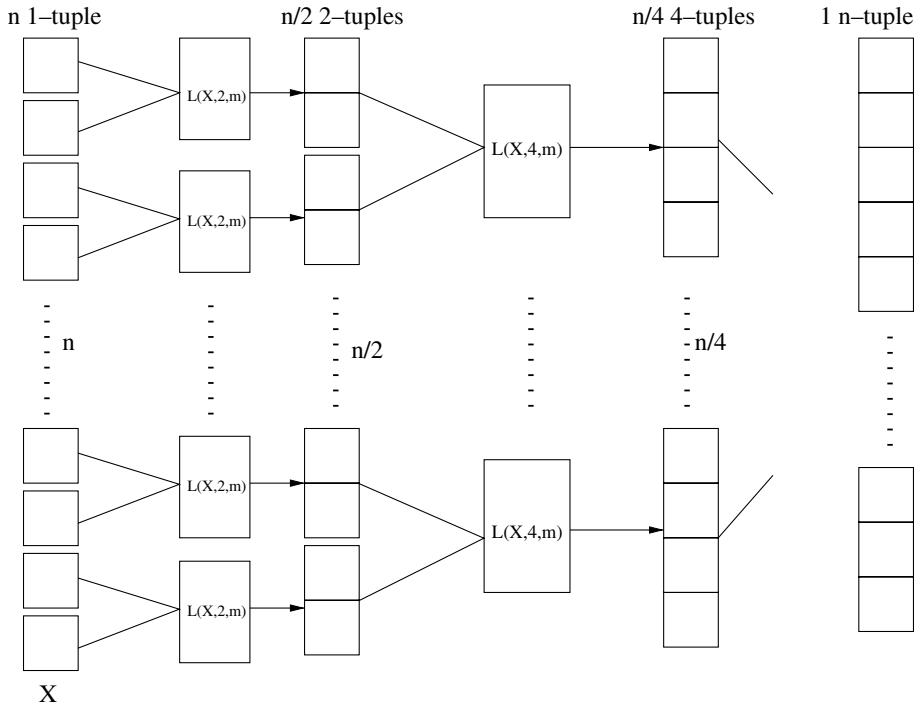

Fig. 1. The quantum gate array to compute $L(X, n, m)$.

algorithm takes the L values of its two children as input and performs the needed multiplication and evaluation of combinatorial and double combinatorial forms involving 2^{i+1} bits each (Lemma 2.1) to get $L(X, 2^{i+1}, m_{i+1})$. An important thing to note here is that we get away with the effort of calculating the m_i (the number of ones) values at each ith step by just adding the m_{i-1} values of the two children and storing it in the work registers M. At the end of recursion, register L has $L(X, n, m)$. If $m = 0$ then $L(x, n, m) = X$ and $S'(X) = X$, so the register L already has the correct value of $S'X$). Otherwise, if $m \geq 1$ we perform the following final additional operation to compute $S'(X)$:

$L \leftarrow L + \sigma_{n,m} + 1$.

This will give us the state $(X, S'(X), 0)$. By the help of the well known Lemma 4.1 stated in Section 4, we can get the desired state $(S'(X), 0, 0)$ with no additional time or storage penalty.

3.2. Analysis

Here, we obtain the time and space bounds of our binary tree implementation of the compression algorithm. As shown below, the time requirements of the unitary operations needed to perform the computation is same as stated in Section 2.

Time complexity. In the binary tree implementation described earlier, as we go up the recursion tree, at each level i we perform $n/2^i$ combinatorial sums and double combinatorial sums (as in Eq. (2.1)) involving 2^i bits each. As we will see later in Section 4, the reversible computation of combinatorial sums and double combinatorial sums involving n bits is $O(M(n \log n) \log n)$ boolean operations where M is the complexity of multiplying two numbers having n bits ($=O(n \log n \log \log n)$). So, the total time complexity of computation of $L(X, m, n)$ in this binary tree organization is:

$\sum_{i=1}^{\lceil \log n \rceil} \frac{n}{2^i} O(M(m_i \log 2^i) \log m_i)$ (m_i is the maximum number of ones in a 2^i tuple at the ith stage)

$= \sum_{i=1}^{\lceil \log n \rceil} \frac{n}{2^i} O(M(im_i) \log m_i)$

$\leq \sum_{i=1}^{\lceil \log n \rceil} \frac{n}{2^i} O(im_i \log(m_i i) \log \log(m_i i) \log m_i)$

$\leq \sum_{i=1}^{\lceil \log n \rceil} nO(i^3 \log i)$ (since $m_i \leq 2^i$)

$\leq O(n(\log^4 n) \log \log n)$

The final operation $L \leftarrow L + \sigma_{n,m} + 1$ used to compute $S'(X)$ when $m \geq 1$ also needs $O(n \log^{O(1)} n)$ operations (which is essentially the time taken to compute $\sigma_{n,m}$). So overall time complexity is given by $O(n(\log^4 n) \log \log n)$.

Space complexity. It is shown in Cleve et al. [10] that the space required to perform multiplication of two numbers of n qubits each, requires $3n$ ($= O(n)$) qubit auxiliary registers. So, the overall auxiliary storage requirement for the above gate array implementation has the following form:

$\sum_{i=1}^{\lceil \log n \rceil} \frac{n}{2^i} O(m_i \log 2^i)$ (m_i is the maximum number of ones in a 2^i tuple at the ith stage)

$= \sum_{i=1}^{\lceil \log n \rceil} \frac{n}{2^i} O(im_i)$

$\leq \sum_{i=1}^{\lceil \log n \rceil} nO(i)$ (since $m_i \leq 2^i$) $\leq O(n(\log^2 n))$

3.3. Quantum decompression via reverse computation of $S'(X)$

For reverse computation, we start with an n-bit number S', the output of $S'(X)$ and then using unitary operations recover the original input X.

Recall that for the special case where $n = 1$, $S'(X) = X$ is the identity map, and so in this case the reverse computation is also just the identity map. Also, recall that if X is an n-bit number with all zeros with $m = 0$, then so is $S'(X)$, and hence the reverse computation is also the identity map if S' a n-bit number with all zeros.

In the following, we assume without loss of generality that $n > 1$ and $m \geq 1$. We will determine the number of ones m in X using the subroutine FINDM(S'). Recall that the number of n bit numbers with $< m$ ones is $1 + \sigma_{n,m}$ where $\sigma_{n,m} = \sum_{i=1}^{m-1} \binom{n}{i}$. Hence, m is the largest number such that $S' > \sigma_{n,m}$.

Algorithm 2 Subroutine FINDM(S')

$m \leftarrow 0$
$\sigma \leftarrow 1$
while $S' > \sigma$ **do**
$m \leftarrow m + 1$
$\sigma \leftarrow \sigma + \binom{n}{m}$
end do

After determining m, we compute $\sigma_{n,m}$ and then determine L by the assignment $L \leftarrow S' - \sigma_{n,m} - 1$. We store L as a signed integer. Recall that our recursive Algorithm 1 for $L(X, n, m)$ terminates leaving the register L with the value $L(X, m, n)$. By Lemma 2.1, we can recursively get the positions of ones in the original input X. The reverse computation is just the same as in the forward direction to compute the desired compression function $S'(X)$ with the difference that now we follow the top down approach in a binary tree to compute from S' the desired value X. That is, we have the initial input as $L(X, m, n)$, then we recursively determine $L(X, m', n')$ and $L(X, m'', n'')$ for each of the $n/2^i$ tuple in stage i after dividing the input into 2^i sets of $n/2^i$ bits at the ith stage. At the bottom of the recursion tree (at the end of $O(\log n)$ stages), we have determined the n single bits giving the actual value of X.

The time and space complexity analysis remains exactly the same as it was in the earlier case for determining $S'(X)$ given that we have the reversible computational complexity of all the subroutines outlined in Section 4. Also, as far as the architecture is considered, it is obvious that we can still follow the binary tree structure as used in the computation of $L(X, n, m)$.

Each operation in our $O(n(\log^4 n) \log\log n)$ time reversible computation of $S'(X)$ consists of certain conditional boolean and arithmetic operations. The conditional boolean operations suffice to be Toffoli gates [26] which take in 3 boolean inputs and negates the first input iff the next two bits are 1. The conditional arithmetic operations suffice to be n-bit (signed) integer negation, addition and multiplication (conditional on a boolean register).

In the next section, we describe the reversible computations of the arithmetic and combinatorial subroutines required for evaluating $L(X, n, m)$ in the binary tree gate array.

4. Reversible computation of functions required in the quantum compression and decompression

Reversible computations are computations where each state transformation is a reversible function, so that any computation can be reversed without loss of information. Landauer [18] showed that irreversible computations must be exothermic in the computing process, and that reversible computations have the property that if executed slowly enough, they (in the limit) can consume no energy in an adiabatic computation. Bennett [3] (also see Bennett and Landauer [4], Landauer [19], Toffoli [26]) showed that any computing machine (e.g., an abstract machine such as a Turing Machine) can be transformed to do reversible computations. Bennett's reversibility construction required extra space to store information to insure reversibility.

Below, we outline some of the basic reversible computations necessary for our compression algorithm. These operations can be performed without preserving any input registers as a part of the output [27].

4.1. Reversible computation of arithmetic functions

- **Reversible addition.** Vedral et al. [27] Given N bit numbers x, y, we can reversibly compute the function: $(x, y) \rightarrow (x, x + y)$. This can be computed in $O(N)$ reversible steps by use of the usual sequential carry-added algorithm. Cleve and DiVincenzo [10] describes in detail how to execute conditional boolean operations in $O(1)$ elementary unitary operations, as well as certain conditional arithmetic operations in $O(n)$ elementary unitary operations, including (signed) integer negation as well as conditional addition of a signed integer n bit register with a constant n bit integer. Vedral et al. [27] give (as a subroutine for their efficient implementation

of the quantum factoring algorithm of Shor [23,24]) a quantum algorithm for addition of two signed integer n bit registers in $O(n)$ elementary unitary operations.

- **Reversible multiplication.** Vedral et al. [27] give a method for the reversible modulo N multiplication of two N bit numbers. In other words, they show how to bring about the transformation $(x, y) \rightarrow (x, xy \bmod N)$ reversibly.
  We use the Schonhagen–Strassen multiplication algorithm Knuth [17] to do n bit integer multiplication in $O(n \log n \log\log n)$ reversible steps. In the quantum gate model, we can do n bit integer multiplication in the same asymptotic steps, again employing the Schonhagen–Strassen multiplication algorithm. (Zalka [28] also gives, again as a subroutine for an efficient implementation of Shor's quantum factoring algorithm, a FFT-based quantum algorithm for multiplication of two n bit integers). For n bit multiplication using the Schonhagen–Strassen multiplication algorithm, the n bits are subdivided into $n' = n/s$ groups of size $s = O(n^{1/2})$. The Schonhagen–Strassen multiplication algorithm requires an n'-point discrete Fourier transform which can be quantum computed in $O(n)$ additions of s bit integers, thus costing a total of $O(n \log n') = O(n \log n)$ elementary unitary operations by the above mentioned quantum addition algorithm. Then the convolution theorem is applied which reduces the problem to the n' recursive multiplications (as given in Vedral et al. [27]) on $O(s)$ bit integers. This requires $O(\log\log n)$ recursive stages. Hence, in the quantum gate model, we can do n bit integer multiplication in $O(n \log n \log\log n)$ elementary unitary operations.

4.2. Bijection of reversible functions

To compute a bijective function $f(x)$ reversibly (like the Schumacher compression), we require that we do not retain any record of the initial state in the output, nor the state of the work bits (since these are completely deducible from the output). These restrictions would seem to make a bijective reversible function $f(x)$ difficult to compute. However, this difficulty can be resolved by an innovative technique due to Bennett [3,5], as follows: It is convenient to first derive a preliminary algorithm PA that retains a record of the initial state and may also make use of temporary work storage. Let us assume that X, L, W are registers and their values are indicated by a tuple (X, L, W). Suppose the input state is given as $(x, 0, 0)$. Then the preliminary algorithm PA provides the mapping $(x, 0, 0) \rightarrow (x, f(x), w)$, where w are the contents of the work registers. We can easily erase these work registers, thus providing a mapping $(x, 0, 0) \rightarrow (x, f(x), 0)$. Then we can apply a known trick to eliminate the initial state; in particular, Bennett [3,5] shows if we have an auxiliary algorithm RA that computes the inverse f^{-1} of the function f, then we can provide the mapping: $(x, y, 0) \rightarrow (x \oplus f^{-1}(y), y, 0)$. Since $x \oplus f^{-1}(f(x)) = x \oplus x = 0$, it follows that RA provides the mapping: $(x, f(x), 0) \rightarrow (0, f(x), 0)$. Finally, an exchange of registers gives the mapping: $(0, f(x), 0) \rightarrow (f(x), 0, 0)$. (Note that it is essential that we also can efficiently compute the inverse f^{-1} of the function f.) Hence, we have from Bennett [3,5] :

Lemma 4.1. *Given a bijective function f, suppose we can reversibly compute in time $T(x)$ a bijective function f and its inverse f^{-1} without preserving input registers as a part of the output. Then in time $O(T(n))$ we can also reversibly compute the bijective mapping: $(x, 0, 0) \rightarrow (f(x), 0, 0)$ without storing x as a part of the output.*

4.3. Reversible computation of combinatorial functions

Reversible computation of factorial.

Lemma 4.2. *Given a number x, we reversibly compute the function: $x \rightarrow x!$. Let N be the number of bits of $x!$. Factorial can be reversibly computed in $O(M(N) \log N) = O(N \log^2 N \log\log N)$ computations*

Proof. We take as input $y = x!$ and let $N = \lceil \log y \rceil$ be the number of bits of y. We apply the Sterling formula to approximate: $y = x! = x^x e^{-x} \sqrt{2\pi x}(1 + O(1/x))$, and taking logarithms we obtain:

$$\begin{aligned} \log y &= \log(x^x e^{-x} \sqrt{2\pi x}(1 + o(1))), \\ &= x \log(x/e) - \frac{1}{2} \log(2\pi x) + O(1). \end{aligned}$$

Let $f(x) = x\log(x/e) - \frac{1}{2}\log(2\pi x)$ and let its inverse function be $f^{-1}(y)$. The inverse $\tilde{x} = f^{-1}(y)$ can be approximately computed up to the required $\log x$ bits in time $O(N\log^2 N\log\log N)$ by the Newton iteration methods of Brent and Kung (see Knuth [17]), and this computation can easily be made reversible. Hence, we compute the factorial of $\tilde{x}+i$ for each i where $|i| \le O(1)$. We output that $x = \tilde{x}+i$ such that $(\tilde{x}+i)! = y$. This has cost $O(1)$ times the cost $O(N\log^2 N\log\log N)$ of the forward computation of factorial, and hence has cost $O(N\log^2 N\log\log N)$. □

Hence by Lemma 4.1, we can reversibly compute factorial in time $O(M(N)\log N) = O(N\log^2 N\log\log N)$ without preserving input registers to form a part of the output.

Reversible computation of combinatorial forms. We provide the Lemmas on the reversible computation of combinatorial and double combinatorial forms necessary for the computation of $S'(X)$.

Lemma 4.3. *Given n bit number with x ones, we can reversibly compute the function*: $x \to \binom{n}{x} = \frac{n!}{(n-x)!}$ *in* $O(M(N)\log N) = O(N\log^2 N\log\log N)$ *steps where N is the number of bits of* $\binom{n}{x}$.

Proof. The forward computation can be easily achieved using a reduction to reversible factorial computation as described above. To reverse the computation of $\binom{n}{x}$, we take as input $y = \binom{n}{x}$ and let $N = \lceil \log y \rceil$ be the number of bits of y. Setting $p = \frac{x}{n}$, we can apply the Sterling formula to approximate, for $x \to \infty$ and $n - x \to \infty$:

$$y = \binom{n}{x} = (p^p(1-p)^{1-p})^{-n}(2\pi p(1-p)n)^{-1/2}(1 + O(1/x)).$$

Taking logarithms we obtain:

$$\begin{aligned}\log y &= -n\log(p^p(1-p)^{1-p}) - \frac{1}{2}\log(2\pi p(1-p)n) + O(1/x),\\ &= -n(p\log(p) + 1 - p\log(1-p)) - \frac{1}{2}\log(2\pi p(1-p)n) + O(1).\end{aligned}$$

Let $g(p) = -n(p\log(p) + 1 - p\log(1-p)) - \frac{1}{2}\log(2\pi p(1-p)n)$ and let its inverse function be $g^{-1}(y)$. The inverse $\tilde{x} = ng^{-1}(y)$ can again be approximately computed up to the required number of bits in time $O(N\log^2 N\log\log N)$ by the methods of Brent [17], and this computation can be made reversible. Hence, we compute $\binom{n}{\tilde{x}+i}$ for each i where $|i| \le O(1)$. We output that $x = \tilde{x} + i$ such that $\binom{n}{\tilde{x}+i} = y$. This has cost $O(1)$ times the cost $O(N\log^2 N\log\log N)$ of the forward computation of $\binom{n}{x}$, and hence has cost $O(N\log^2 N\log\log N)$. Hence by Lemma 4.1, we can reversibly compute $\binom{n}{x}$ in time $O(M(N)\log N) = O(N\log^2 N\log\log N)$ without preserving input registers as a part of the output. □

Lemma 4.4. *The combinatorial sum* $\sigma_{n,m} = \sum_{i=1}^{m-1}\binom{n}{i}$ *and the double combinatorial sum* $\sigma'_{n,m,m'} = \sum_{i=1}^{m'-1}\binom{n'}{i}\binom{n''}{m-i}$ *can be solved in* $O(M(m\log n)\log m)$ *reversible boolean operations, where* $M(x)$ *is the time complexity of multiplying two numbers with x bits.*

Proof. Here, we give the proofs separately for different cases.

(1) **Computation of sums of combinatorial forms** $\left(\sigma_{n,m} = \sum_{i=1}^{m-1}\binom{n}{i}\right)$**:** Since $\binom{n}{m}$ is a product of $O(m)$ numbers each with $O(\log n)$ bits, it follows that $\binom{n}{m}$ is an $O(m\log n)$ bit number. So $\sigma_{n,m}$ is a sum of

n numbers each of $O(m \log n)$ bits, and so is also an $O(m \log n)$ bit number. Note that since $\binom{n}{i} = \binom{n}{i-1}(n-(i-1))$, it follows that $\sigma_{n,m}$ has a recursive expansion:

$$\sigma_{n,m} = n(1 + (n-1)(1 + \ldots (n-(m-3))(1 + (n-(m-2))) \ldots).$$

To compute $\sigma_{n,m}$, we will apply a divide and conquer of this recursive expansion. For $1 \leq a \leq b \leq n$, let $\tau_{n,a,b} = \left(\sum_{i=a}^{b} \binom{n}{i}\right) \Big/ \binom{n}{a-1}$. Note that by definition, $\sigma_{n,m} = \tau_{n,1,m-1}$. Observe that $\tau_{n,a,b}$ has the recursive expansion:

$$\begin{aligned}
\tau_{n,a,b} &= \sum_{i=a}^{b} \frac{(n-(a-1))!}{(n-i)!}, \\
&= \sum_{i=a}^{b} (n-(a-1))(n-a)\ldots(n-(i-2))(n-(i-1)), \\
&= (n-(a-1))(1 + (n-a)(1 + \ldots (n-(b-2)) \\
&\quad (1 + (n-(b-1))) \ldots).
\end{aligned}$$

To compute $\tau_{n,a,b}$, we apply divide and conquer of the expansion at $a' = \lfloor \frac{b-a}{2} \rfloor$, using the identity $\tau_{n,a,b} = \tau_{n,a,a'} + c_{n,a,a'}\tau_{n,a',b}$, where

$$\begin{aligned}
c_{n,a,a'} &= \prod_{i=a}^{a'-1} (n-(i-1)), \\
&= (n-(a'-2))(n-(a'-3))\ldots(n-a)(n-(a-1)).
\end{aligned}$$

Since $c_{n,a,a'}$ is a product of $O(b-a)$ numbers each with $O(\log n)$ bits, it follows that $c_{n,a,a'}$ is an $O((b-a)\log n)$ bit number. Let $M(N)$ be the bit complexity of multiplying two N bit numbers. Hence the time cost for the recursive computation of $\tau_{n,a,b}$, for $m = b-a$, is $T(n,m) \leq O(M(m \log n)) + T(n, \lfloor \frac{m}{2} \rfloor) + T(n, \lceil \frac{m}{2} \rceil) \leq O(M(m \log n)) + 2T(n, \lceil \frac{m}{2} \rceil)$. This recurrence is bounded as $T(n,m) \leq O(M(m \log n) \log m)$. Hence $\sigma_{n,m}$ also costs $O(M(m \log n) \log m)$ boolean operations.

(2) **Inverse and reversible computation of sums of combinatorial forms:** Let $F(m) = \sigma_{n,m}$ for fixed n. Since the sum of combinatorial forms is dominated by its highest term, the inverse $F^{-1}(y) = m$ of the function $F(m)$ can also be efficiently be reversibly computed in time $O(M(m \log n) \log m)$ by techniques similar to those described above for the inverse of $\binom{n}{x}$. Hence, we can reversibly compute $\sigma_{n,m}$ in $O(M(m \log n) \log m)$ boolean operations without preserving input registers as a part of the output.

(3) **Computation of sums σ' of double combinatorial forms:** Here, we compute $\sigma'_{n,m,m'} = \sum_{i=1}^{m'-1} \binom{n'}{i}\binom{n''}{m-i}$ which is the number of n bit binary numbers with at least 1 ones and at most $m'-1$ ones in the first $n' = \lfloor n/2 \rfloor$ bits and $n'' = n - n'$. Note that

$$\binom{n'}{i} = \binom{n'}{i-1}(n'-(i-1))$$

and

$$\binom{n''}{m-i} = \binom{n''}{m-(i-1)}(n''-(m-(i-1))),$$

so,

$$\binom{n'}{i}\binom{n''}{m-i} = \binom{n'}{i-1}\binom{n''}{m-(i-1)}(n'-(i-1))(n''-(m-(i-1))).$$

It follows that $\sigma'_{n,m,m'}$ has a recursive expansion:

$$\begin{aligned}\sigma'_{n,m,m'} = {} & \binom{n''}{m-2}(n'(n''-m))(1+((n'-1)(n''-(m-1)))\\ & (1+\ldots((n'-(m'-3))(n''-(m-(m'-3))))(1+((n'-(m'-2))\\ & (n''-(m-(m'-2)))))\ldots).\end{aligned}$$

To compute $\sigma'_{n,m,m'}$, we will apply a divide and conquer of this recursive expansion. For $1 \le a \le b \le n$, let:

$$\tau'_{n,m,a,b} = \left(\sum_{i=a}^{b}\binom{n'}{i}\binom{n''}{m-i}\right) \Big/ \left(\binom{n'}{a-1}\binom{n''}{m-(a-1)}\right).$$

Note that by definition, $\sigma'_{n,m,m'} = \tau'_{n,m,1,m'-1}$. Observe that $\tau'_{n,m,a,b}$ has the recursive expansion:

$$\begin{aligned}\tau'_{n,m,a,b} = {} & ((n'-(a-1))(n''-(m-(a-1))))\\ & (1+((n'-a)(n''-(m-a)))\\ & (1+\ldots((n'-(b-2))(n''-(m-(b-2))))\\ & (1+((n'-(b-1))(n''-(m-(b-1)))))\ldots).\end{aligned}$$

To compute $\tau'_{n,a,b}$, we apply divide and conquer of the expansion at $a' = \lfloor \frac{b-a}{2} \rfloor$, using the identity $\tau'_{n,m,a,b} = \tau'_{n,m,a,a'} + c'_{n,m,a,a'}\tau'_{n,m,a',b}$, where:

$$\begin{aligned}c'_{n,m,a,a'} &= \prod_{i=a}^{a'-1}(n'-(i-1))(n''-(m-(i-1))),\\ &= (n'-(a'-2))(n''-(m-(a'-2)))\\ &\quad (n-(a'-3))(n''-(m-(a'-3)))\ldots\\ &\quad (n'-a)(n''-(m-a))(n'-(a-1))(n''-(m-(a-1))).\end{aligned}$$

Since $c'_{n,a,a'}$ is a product of $O((b-a))$ numbers each with $O(\log n)$ bits, it follows that $c'_{n,a,a'}$ is an $O((b-a)\log n)$ bit number. Recall that $\sigma'_{n,m,m'}$ is a product of $O(m)$ numbers each with $O(\log n)$ bits, so it follows that $\sigma'_{n,m,m'}$ is an $O(m\log n)$ bit number. Hence, the time cost for the recursive computation of $\tau_{n,a,b}$ is $T'(n,m) \le O(M(m\log n)) + T(n, \lfloor \frac{m}{2} \rfloor) + T'(n, \lceil \frac{m}{2} \rceil) \le O(M(m\log n)) + 2T'(n, \lceil \frac{m}{2} \rceil)$. This recurrence is bounded as $T'(n,m) \le O(M(m\log n)\log m)$. Hence $\sigma'_{n,m,m'}$ can be recursively evaluated in $O(M(m\log n)\log m)$ boolean operations.

(4) **Inverse and reversible computation of sums of double combinatorial forms:** Let $y = G(m,m') = \sigma'_{n,m,m'}$ for a fixed n. Since the sum of double combinatorial forms is dominated by its highest term, the inverse $G^{-1}(y) = (m,m')$ of the function $y = G(m,m')$ can also be efficiently be reversibly computed in $O(M(m\log n)\log m)$ boolean operations by techniques similar to those described above for the inverse of $\binom{n}{x}$. □

So, all the above operations used in the computation of $S'(X)$ can be reversibly computed without preserving the input as a part of the output. These computations can be done by the use of efficient quantum algorithms [10] using basic quantum gates and reversible arithmetic and conditional operations.

Hence by Lemma 4.1, we can reversibly compute the arithmetic, combinatorial and double combinatorial operations used in the gate array computation of $S'(X)$ without preserving the input registers as a part of the input.

In Sections 2 and 3, we described how to do the reversible computation of $S'(X, n, m)$ in $\leq O(n(\log^4 n)\log\log n)$ deterministic boolean steps using the binary tree gate array model. In this section, we have shown how to reversibly compute each of the subroutines required in the $S'(X, n, m)$ computation. Hence, we obtain the main result:

Theorem 4.1. *The overall transformation* (*for all the states*) $S'(X)$ *be done as a quantum computation using only elementary unitary quantum operations*, *with asymptotic number of* $O(n(\log^4 n)\log\log n)$ *steps*.

5. Conclusion

We give an efficient deterministic algorithm for a modified Schumacher compression encoding function (with asymptotically optimal size and fidelity) using $O(n(\log^4 n)\log\log n)$ boolean operations. To achieve our goal we have also shown how the various subroutines required by our algorithm can be made reversible, and the modified Schumacher encoding and decoding can be efficiently computed by a quantum computer in $O(n(\log^4 n)\log\log n)$ elementary unitary steps. We exploit the inherent tree structure of the divide and conquer algorithm to obtain the gate array to evaluate the $S'(X)$ in asymptotically the same number $O(n(\log^4 n)\log\log n)$ of operations as required by our reversible algorithm for our modified Schumacher encoding function.

Acknowledgments

The authors thank the reviewer(s) of this paper for their valuable comments and suggestions in improving the manuscript.

References

[1] H. Barnum, C.A. Fuchs, R. Jozsa, B. Schumacher, Phys. Rev. A 54 (1996) 4707–4711.
[2] A. Barenco, C.H. Bennett, R. Cleve, D.P. DiVincenzo, N. Margolus, P. Shor, T. Sleator, J. Smolin, H. Weinfurter, Elementary gates for quantum computation, Phys. Rev. A 52 (1995) 3457.
[3] C.H. Bennett, Logical reversibility of computation, IBM J. Res. Develop. 17 (1973) 525.
[4] C.H. Bennett, R. Landauer, Physical limits of computation, Sci. Am. (1985) 48.
[5] C.H. Bennett, Time/space trade-offs for reversible computation, SIAM J. Comput. 18 (1989) 766.
[6] C.H. Bennett, Quantum information and computation, Phys. Today (1995) 24–30.
[7] C.H. Bennett, G. Brassard, R. Jozsa, D. Mayers, A. Peres, B. Schumacher, W. Wootters, Reduction of quantum entropy by reversible extraction of classical information, J. Mod. Opt. (1994).
[8] S.L. Braustein, C.A. Fuchs, D. Gottesman, H.-K. Lo, A quantum analog of Huffman Coding, IEEE Trans. Inf. Theory 46 (2000) 1644–1649.
[9] T.M. Cover, J.A. Thomas, Elements of Information Theory, Wiley and Sons, New York, 1991.
[10] R. Cleve, D.P. DiVincenzo, Schumacher's quantum data compression as a quantum computation, Phys. Rev. A 54 (1996) 2636–2650.
[11] D. Deutsch, Quantum computational network, Proceedings of the Royal Society, London 425 (1989) 73–90.
[12] D.P. DiVincenzo, Two-bit gates are universal for quantum computation, Phys. Rev. A 51 (1995) 1015.
[13] C. Fuchs, C. Caves, Ensemble-dependent bounds for accessible information in quantum mechanics, Phys. Rev. Lett. 73 (1994) 3047–3050.
[14] A. Holevo, Some estimates of the information transmitted by quantum communication channels, Probl. Inf. Transm. 9 (1973) 177–183.
[15] R. Jozsa, B. Schumacher, A new proof of the quantum noiseless coding theorem, J. Mod. Opt. 41 (1994) 2343–2349.
[16] R. Jozsa, M. Horodecki, P. Horodecki, R. Horodecki, Universal quantum data compression, Phys. Rev. Lett. 81 (1998) 1714–1718.
[17] D. Knuth, in: The Art of Computer Programming, Seminumerical Algorithms, vol. II, Addison-Wesley, 1973.
[18] R. Landauer, Irreversibility and heat generation in the computing process, IBM J. Res. Dev. 5 (1961) 183.
[19] R. Landauer, Is quantum mechanics useful?, Proc. Roy. Soc. Lond. (1996).
[20] J. von Neumann, Mathematical Foundations of Quantum Mechanics, Springer Verlag, 1932.

[21] J.H. Reif, On the Impossibility of Interaction-Free Sensing for Small I/O Bandwidth (Online preprint at: http://www.cs.duke.edu/~reif/paper/qsense/qsense.ps).
[22] B. Schumacher, Quantum coding, Phys. Rev. Lett. A 51 (1995) 2738–2747.
[23] P.W. Shor, Algorithms for quantum computation: discrete logarithms and factoring, Proceedings of the 35th Annual Symposium on Foundations of Computer Science, Santa Fe, NM, 1994.
[24] P.W. Shor, Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer, SIAM J. Comput. 26 (1997) 1484.
[25] K.Y. Szeto, Data compression of quantum code, ArXiv Quantum Physics e-prints (1996).
[26] T. Toffoli, Reversible Computing, in: Automata Languages and Programming, Springer, New York, 1980, pp. 632.
[27] V. Vedral, A. Barenco, A. Ekert, Quantum networks for elementary arithmetic operations, Phys. Rev. A 54 (1995) 147–153.
[28] C. Zalka, Fast versions of Shor's quantum factoring algorithm, LANL e-print quant-ph/9806084, 1998.