Quantum Information and Computation, Vol. 13, No. 11&12 (2013) 0937–0962
© Rinton Press

A 2D NEAREST-NEIGHBOR QUANTUM ARCHITECTURE FOR FACTORING IN POLYLOGARITHMIC DEPTH

PAUL PHAM[a]
Quantum Theory Group, University of Washington, Box 352350,
Seattle, WA 98195, USA

KRYSTA M. SVORE[b]
Quantum Architectures and Computation Group, Microsoft Research, One Microsoft Way,
Redmond, WA 98052, USA

Received October 11, 2012
Revised March 29, 2013

We present a 2D nearest-neighbor quantum architecture for Shor's algorithm to factor an n-bit number in $O(\log^3 n)$ depth. Our implementation uses parallel phase estimation, constant-depth fanout and teleportation, and constant-depth carry-save modular addition. We derive upper bounds on the circuit resources of our architecture under a new 2D model which allows a classical controller and parallel, communicating modules. We provide a comparison to all previous nearest-neighbor factoring implementations. Our circuit results in an exponential improvement in nearest-neighbor circuit depth at the cost of a polynomial increase in circuit size and width.

Keywords: quantum architecture, prime factorization, Shor's algorithm, nearest-neighbor, carry-save addition

Communicated by: B Kane & M Mosca

1 Introduction

Shor's factoring algorithm is a central result in quantum computing, with an exponential speed-up over the best known classical algorithm [1]. As the most notable example of a quantum-classical complexity separation, much effort has been devoted to implementations of factoring on a realistic architectural model of a quantum computer [2, 3, 4, 5, 6]. We can bridge the gap between the theoretical algorithm and a physical implementation by describing the layout and interactions of qubits at an intermediate, architectural level of abstraction. This gives us a model for measuring circuit resources and their tradeoffs. In this work, we present a circuit implementation for prime factorization of an n-bit integer on a two-dimensional architecture that allows concurrent (parallel) two-qubit operations between neighboring qubits, an omnipresent classical controller, and modules which are allowed to teleport qubits to each other. We call this new model 2D CCNTCM. We show that our circuit construction is asymptotically more efficient in circuit depth than previous state-of-the-art techniques for

[a] ppham@cs.washington.edu
[b] ksvore@microsoft.com

nearest-neighbor architectures, achieving a depth of $O(\log^3 n)$, a size of $O(n^4 \log n)$, and a width of $O(n^4)$ qubits, as detailed in Table 3 of Section 8.

Our technique hinges on several key building blocks. Section 2 introduces quantum architectural models, circuit resources, and constant-depth communication techniques due to [7, 8]. Section 3 places our work in the context of existing results. In Section 4, we provide a self-contained pedagogical review of the carry-save technique and encoding. In Section 5 we modify and extend the carry-save technique to a 2D modular adder, which we then use as a basis for a modular multiplier (Section 6) and a modular exponentiator (Section 7). For each building block, we provide numerical upper bounds for the required circuit resources. Finally, we compare our asymptotic circuit resource usage with other factoring implementations.

2 Background

Quantum architecture is the design of physical qubit layouts and their allowed interactions to execute quantum algorithms efficiently in time, space, and other resources. In this paper, we focus on designing a realistic nearest-neighbor circuit for running Shor's factoring algorithm on two-dimensional architectural models of a physical quantum device with nearest-neighbor interactions.

2.1 Architectural Models and Circuit Resources

Following Van Meter and Itoh [5], we distinguish between a model and an architectural implementation as follows. A *model* is a set of constraints and rules for the placement and interaction of qubits. An *architecture* (or interchangeably, an *implementation* or a *circuit*) is a particular spatial layout of qubits (as a graph of vertices) and their constrained interactions (edges between the vertices), following the constraints of a given model. In this section, we describe several models which try to incorporate resources of physical interest from experimental work. We also introduce a new model, 2D CCNTCM, which we will use to analyze our current circuit.

The most general model is called Abstract Concurrent (AC) and allows arbitrary, long-range interactions between any qubits and concurrent operation of quantum gates. This corresponds to a complete graph with an edge between every pair of nodes, and is the model assumed in most quantum algorithms.

A more specialized model restricts interactions to nearest-neighbor, two-qubit, concurrent gates (NTC) in a regular one-dimensional chain (1D NTC), which is sometimes called linear nearest-neighbor (LNN). This corresponds to a line graph. This is a more realistic model than AC, but correspondingly, circuits in this model may incur greater resource overheads.

To relieve movement congestion, we can consider a two-dimensional regular grid (2D NTC), where each qubit has four planar neighbors and there is an extra degree of freedom in which to move data. In this paper, we extend the 2D NTC model in three ways. The first two extensions are described in Section 2.2, and the third extension is described in Section 2.3.

2.2 2D CCNTC*: Two-Dimensional Nearest-Neighbor Two-Qubit Concurrent Gates with Classical Controller*

The first extension allows arbitrary planar graphs with bounded degree, rather than a regular square lattice. Namely, we assume qubits lie in a plane and edges are not allowed to intersect.

All qubits are accessible from above or below by control and measurement apparatus. Whereas 2D NTC conventionally assumes each qubit has four neighbors, we consider up to six neighbors in a roughly hexagonal layout. The second extension is the realistic assumption that classical control (CC) can access every qubit in parallel, and we do not count these classical resources in our implementation since they are polynomially bounded. The classical controllers correspond to fast digital computers which are available in actual experiments and are necessary for constant-depth communication in the next section.

We call an AC or NTC model augmented by these two extensions CCAC and CCNTC, respectively. Before we describe the third extension, let us formalize our model for 2D CC-NTC, with definitions that are (asymptotically) equivalent to those in [8].

Definition 1 *2D CCNTC architecture consists of*

- *a quantum computer QC which is represented by a planar graph (V, E). A node $v \in V$ represents a qubit which is acted upon in a circuit, and an undirected edge $(u, v) \in E$ represents an allowed two-qubit interaction between qubits $u, v \in V$. Each node has degree at most* 6*.*
- *a universal gate set $\mathcal{G} = \{X, Z, H, T, T^\dagger, CNOT, MeasureZ\}$.*
- *a deterministic machine (classical controller) CC that applies a sequence of concurrent gates in each of D timesteps.*
- *In timestep i, CC applies a set of gates $G_i = \{g_{i,j} \in \mathcal{G}\}$. Each $g_{i,j}$ operates in one of the following two ways:*

  1. *It is a single-qubit gate from $\mathcal{G}$ acting on a single qubit $v_{i,j} \in V$*
  2. *It is the gate CNOT from $\mathcal{G}$ acting on two qubits $v_{i,j}^{(1)}, v_{i,j}^{(2)} \in V$ where $(v_{i,j}^{(1)}, v_{i,j}^{(2)}) \in E$*

  All the $g_{i,j}$ can only operate on disjoint qubits for a given timestep i. We define the support of G_i as V_i, the set of all qubits acted upon during timestep i.

$$V_i = \bigcup_{j : g_{i,j} \in G_i} v_{i,j} \cup v_{i,j}^{(1)} \cup v_{i,j}^{(2)} \tag{1}$$

We can then define the three conventional circuit resources in this model.

circuit depth (D): the number of concurrent timesteps.

circuit size (S): the total number of non-identity gates applies from $\mathcal{G}$, equal to $\sum_{i=1}^{D} |G_i|$.

circuit width (W): is the total number of qubits operated upon by any gate, including inputs, outputs, and ancillae. It is equal to $|\bigcup_{i=1}^{D} V_i|$.

We observe that the following relationship holds between the circuit resources. The circuit size is bounded above by the product of circuit depth and circuit width, since in the worst case, every qubit is acted upon by a gate for every timestep of a circuit. The circuit depth is

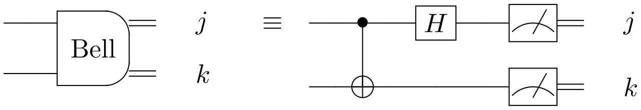

Fig. 1. A circuit for measurement in the Bell state basis.

also bounded above by the size, since in the worst case, every gate is executed serially without any concurrency.

$$D \leq S \leq D \cdot W \tag{2}$$

The set $\mathcal{G}$ includes measurement in the Z basis, which is actually not a unitary operation, or gate per se. All other gates in $\mathcal{G}$ form a universal set of unitary gates [9]. In this paper we will treat the operations in $\mathcal{G}$ as *elementary gates.*

We also find it useful to define a Bell basis measurement using operations from $\mathcal{G}$. A circuit performing this measurement is shown in Figure 1 and has depth 4, size 4, and width 2.

The third extension to our model, and the most significant, is to consider multiple disconnected planar graphs, each of which is a 2D CCNTC architecture. This is described in more detail in the next section.

2.3 2D CCNTCM***: Two-Dimensional Nearest-Neighbor Two-Qubit Concurrent Gates with Classical Controller and Modules***

A single, contiguous 2D lattice which contains an entire quantum architecture which may be prohibitively large to manufacture. In practice, scalable experiments will probably use many smaller quantum computers which communicate by means of shared entanglement [10]. We call these individual machines *modules*, each of which is a self-contained 2D CCNTC lattice. This should not be confused with the word “modular” as in “modular arithmetic” or as referring to the modulus m which we are trying to factor.

We treat these modules and teleportations between them as nodes and edges, respectively, in a higher-level planar graph. The teleportations each transmit one qubit from one module to another, from any location within the source module and to any location within the destination module, making use of the omnipresent classical controller. The modules can be arbitrarily far apart and have arbitrary connectivity with other modules.

A single module can be part of multiple teleportation operations in a single timestep, as long as they involve disjoint qubits within the module. We justify this assumption in that it is possible to establish entanglement between multiple quantum computers in parallel. We call this new model 2D CCNTCM, and we argue that is captures the essential aspects of 2D architectures without being overly sensitive to the exact geometry of the lattices involved. An graphic depiction of three modules in 2D CCNTCM is shown in Figure 2. Each module contains within it a 2D CCNTC lattice. We can equivalently consider the omnipresent, single classical controller as a collection of multiple classical controllers, one for each module or teleportation operation, which can inter-communicate classically and share a clock.

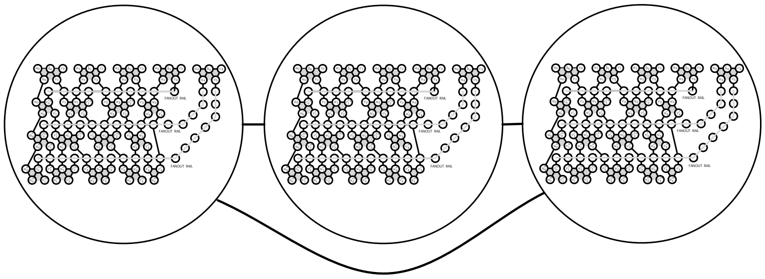

Fig. 2. Three modules in the 2D CCNTCM model

Definition 1 *A* 2D CCNTCM *architecture consists of*

- *a quantum computer $\overline{QC}$ which is represented by a planar graph $(\overline{V}, \overline{E})$. A node $\overline{v} \in \overline{V}$ represents a module, or a graph (V, E) from a* 2D CCNTC *architecture defined previously. It can have unbounded degree. An undirected edge $(\overline{u}, \overline{v}) \in \overline{E}$ represents an allowed teleportation from any qubit in module $\overline{u}$ to another qubit in module $\overline{v}$.*

- *All modules are restricted to be linear in the number of their qubits: $|V| = O(n)$ for all $(V, E) \in \overline{V}$.*

- *a universal gate set $\mathcal{G} = \{X, Z, H, T, T^\dagger, CNOT, MeasureZ\}$ for the qubits* within the same *modules which is the same as for* 2D CCNTC*, and an additional operation $Teleport$ which only operates on qubits* in different *modules.*

- *a deterministic machine (classical controller) $\overline{CC}$ that applies a sequence of concurrent gates in each of $D + \overline{D}$ timesteps. This can be a separate classical controller for every pair of modules.*

- *In timestep i, $\overline{CC}$ applies gates $G_i = \{g_{i,j} : g_{i,j} \in \mathcal{G} \vee g_{i,j} = Teleport\}$. That is, there are two kinds of timesteps with respect to the kinds of gates which operate within them.*

  1. *In the first kind, gates are exclusively from $\mathcal{G}$ operating within modules, as described for* 2D CCNTC *above. We say there are D such timesteps.*
  2. *In the second kind, gates are exclusively $Teleport$ gates between two qubits $v^{(1)}_{i,j} \in \overline{v}_1$ and $v^{(2)}_{i,j} \in \overline{v}_2$ for (possibly non-distinct) modules $\overline{v}_1, \overline{v}_2 \in \overline{V}$. Again, all such qubits much be distinct within a timestep. We say there are $\overline{D}$ such timesteps.*

  Again, we define the support of G_i as V_i, the set of all qubits acted upon by any $g_{i.j}$, which includes all the modules.

$$V_i = \bigcup_{j : g_{i,j} \in G_i} v_{i,j} \cup v^{(1)}_{i,j} \cup v^{(2)}_{i,j} \cup \overline{v}_1 \cup \overline{v}_2 \tag{3}$$

We measure the efficiency of a circuit in this new module using not just the three conventional circuit resources, but with three novel resources based on modules.

module depth ($\overline{D}$): the depth of consecutive teleportations between modules.

module size ($\overline{S}$): the number of total qubits teleported between any two modules over all timesteps.

module width ($\overline{W}$): the number of modules whose qubits are acted upon during any timestep.

We note the following relationship between circuit width and module width.

$$W = O(n\overline{W}) \tag{4}$$

2.4 Circuit Resource Comparisons

Counting gates from $\mathcal{G}$ as having unit size and unit depth is an overestimate compared to the model in [3], in which a two-qubit gate has unit size and unit depth and absorbs the depth and size of any adjacent single-qubit gates. We intend for this more pessimistic estimate to reflect the practical difficulties in compiling these gates using a non-Clifford gate in a fault-tolerant way, such as the T gate or the Toffoli gate [11].

In both our resource counting method and that of [12, 3], multiple gates acting on disjoint qubits can occur in parallel during the same timestep. For each building block, from modular addition to modular multiplication and finally to modular exponentiation, we provide closed form equations upper-bounding the required circuit resources as a function of n, the size of the modulus m to be factored. We will use the term *numerical upper bound* to distinguish these formulae from asymptotic upper bounds.

It is possible to reduce the numerical constants with more detailed analysis, which would be important for any physical implementation. However, we have chosen instead to simplify the number of terms in the formulae for the current work. We do not intend for these upper bounds to represent the optimal or final work in this area.

The modular adder in Section 5 and its carry-save subcomponents only occur within a single module, so we only give their circuit resources in terms of circuit depth, circuit size, and circuit width. For the modular multiplier in Section 6 and the modular exponentiator in Section 7, we also give circuit resources in terms of module depth, module size, and module width.

2.5 Constant-depth Teleportation and Fanout

Communication, namely the *moving* and *copying* of quantum information, in nearest-neighbor quantum architectures is challenging. The first challenge of moving quantum information from one site to another over arbitrarily long distances can be addressed by using the constant-depth teleportation circuit shown in Figure 3, illustrated using standard quantum circuit notation [13]. This requires the circuit resources shown in Table 1. The depth includes a layer of H gates; a layer of CNOTs; an interleaved layer of Bell basis measurements; and two layers of Pauli corrections (X and Z for each qubit), occurring concurrently with resetting the $|j\rangle$ and $|k\rangle$ qubits back to $|0\rangle$. These correction layers are not shown in the circuit.

Although general cloning is impossible [13], the second challenge of copying information can be addressed by performing an unbounded quantum fanout operation: $|x, y_1, \ldots, y_n\rangle \rightarrow |x, y_1 \oplus x, \ldots, y_n \oplus x\rangle$. This is used in our arithmetic circuits when a single qubit needs to control (be entangled with) a large quantum register (called a *fanout rail*). We employ a constant-depth circuit due to insight from measurement-based quantum computing [16] that relies on the creation of an n-qubit cat state [15].

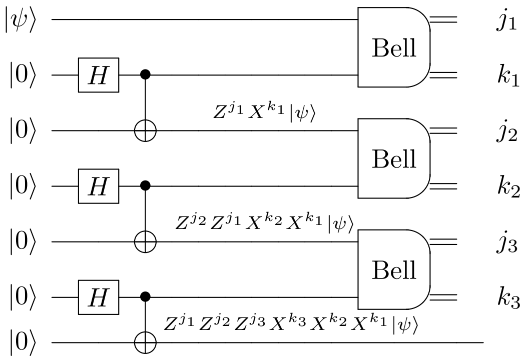

Fig. 3. Constant-depth circuit based on [14, 15] for teleportation over $n = 5$ qubits [8].

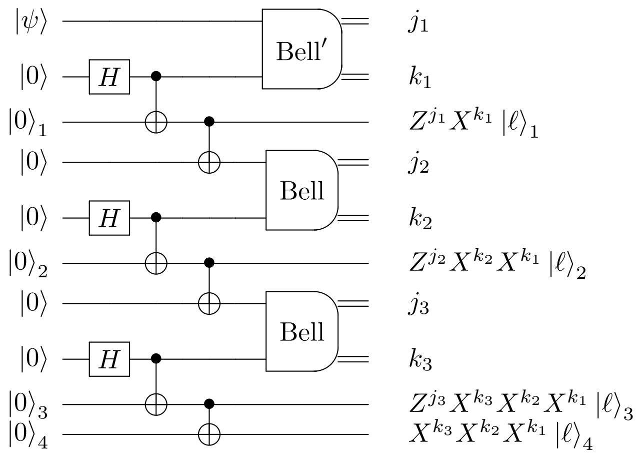

Fig. 4. Constant-depth circuits based on [14, 15] for fanout [7] of one qubit to $n = 4$ entangled copies.

This circuit requires $O(1)$-depth, $O(n)$-size, and $O(n)$-width. Approximately two-thirds of the ancillae are reusable and can be reset to $|0\rangle$ after being measured. Numerical upper bounds are given in Table 1. The constant-depth fanout circuit is shown in Figure 4 for the case of fanning out a given single-qubit state $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ to four qubits. The technique works by creating multiple small cat states of a fixed size (in this case, three qubits), linking them together into a larger cat state of unbounded size with Bell basis measurements, and finally entangling them with the source qubit to be fanned out. The qubits marked $|\ell\rangle$ are entangled into the larger fanned out state given in Equation 5. The Pauli corrections from the cat state creation are denoted by X^{k_2}, X^{k_3}, Z^{j_2} and Z^{j_3} on qubits ending in states $|\ell\rangle_1$, $|\ell\rangle_2$, $|\ell\rangle_3$, and $|\ell\rangle_4$. The Pauli corrections X^{k_1} and Z^{j_1} are from the Bell basis measurement

Circuit Name	Depth	Size	Width
Teleportation from Figure 3	7	3n + 4	n+1
Fanout from Figure 4	9	10n - 9	3n-1
Un-fanout	$8\log_2(2n)$	$33n\log_2(2n) + 10\log_2^2(2n)$	$3n-1$

Table 1. Circuit resources for teleportation, fanout, and un-fanout (consisting of alternating rounds of constant-depth teleportation and CNOT).

entangling the cat state with the source qubit (denoted Bell′).

$$Z_1^{j_1} X_1^{k_1} Z_2^{j_2} X_2^{k_2} X_2^{k_1} Z_3^{j_3} X_3^{k_3} X_3^{k_2} X_3^{k_1} X_4^{k_3} X_4^{k_2} X_4^{k_1} \left(\alpha \left|0\right\rangle_1 \left|0\right\rangle_2 \left|0\right\rangle_3 \left|0\right\rangle_4 + \beta \left|1\right\rangle_1 \left|1\right\rangle_2 \left|1\right\rangle_3 \left|1\right\rangle_4\right) \tag{5}$$

The operators X_i^k and Z_h^j denote Pauli X and Z operators on qubits i and h, controlled by classical bits k and j, respectively. These corrections are enacted by the classical controller based on the Bell measurement outcomes (not depicted). Note the cascading nature of these corrections. There can be up to $n-1$ of these X and Z corrections on the same qubit, which can be simplified by the classical controller to a single X and Z operation and then applied with a circuit of depth 2 and size 2. Also, given the symmetric nature of the cat state, there is an alternate set of Pauli corrections which would give the same state and is of equal size to corrections given above.

Reversing the fanout (un-fanout) in constant depth is an interesting problem. Doing so would allow us to improve the overall depth of our factoring implementation to $O(\log^2 n)$ instead of $O(\log^3 n)$. In this work it is sufficient to perform un-fanout using alternating rounds of teleportation and CNOT among the n fanned-out qubits in a logarithmic-depth binary tree. The resources for this are given in Table 1.

From an experimental perspective, it is physically efficient to create a cat state in trapped ions using the Mølmer-Sørensen gate [17][18]. However, the fanout circuit for the 2D CCNTCM model would still be useful for other technologies, such as superconducting qubits on a two-dimensional lattice.

3 Related Work

Our work builds upon ideas in classical digital and reversible logic and their extension to quantum logic. Any circuit implementation for Shor's algorithm requires a quantum adder. Gossett proposed a quantum algorithm for addition using classical carry-save techniques to add in constant-depth and multiply in logarithmic-depth, with a quadratic cost in qubits (circuit width) [19]. The techniques relies on encoded addition, sometimes called a 3-2 adder, and derives from classical Wallace trees [20].

Takahashi and Kunihiro discovered a linear-depth and linear-size quantum adder using zero ancillae [21]. They also developed an adder with tradeoffs between $O(n/d(n))$ ancillae and $O(d(n))$-depth for $d(n) = \Omega(\log n)$ [22]. Their approach assumes unbounded fanout, which had not previously been mapped to a nearest-neighbor circuit until our present work.

Studies of architectural constraints, namely restriction to a 2D planar layout, were experimentally motivated. For example, these layouts correspond to early ion trap proposals [23] and were later analyzed at the level of physical qubits and error correction in the context of

Shor's algorithm [24]. Choi and Van Meter designed one of the first adders targeted to a 2D architecture and showed it runs in $\Theta(\sqrt{n})$-depth on 2D NTC [25] using $O(n)$-qubits with dedicated, special-purpose areas of a physical circuit layout.

Modular exponentiation is a key component of quantum period-finding (QPF), and its efficiency relies on that of its underlying adder implementation. Since Shor's algorithm is a probabilistic algorithm, multiple rounds of QPF are required to amplify success probability arbitrarily close to 1. It suffices to determine the resources required for a single round of QPF with a fixed, modest success probability (in the current work, 3/4).

The most common approach to QPF performs controlled modular exponentiation followed by an inverse quantum Fourier transform (QFT) [13]. We will call this *serial QPF*, which is used by the following implementations.

Beauregard [2] constructs a cubic-depth quantum period-finder using only $2n + 3$ qubits on AC. It combines the ideas of Draper's transform adder [26], Vedral et al.'s modular arithmetic blocks [27], and a semi-classical QFT. This approach was subsequently adapted to 1D NTC by Fowler, Devitt, and Hollenberg [12] to achieve resource counts for an $O(n^3)$-depth quantum period-finder. Kutin [3] later improved this using an idea from Zalka for approximate multipliers to produce a QPF circuit on 1D NTC in $O(n^2)$-depth. Thus, there is only a constant overhead from Zalka's own factoring implementation on AC, which also has quadratic depth [28]. Takahashi and Kunihiro extended their earlier $O(n)$-depth adder to a factoring circuit in $O(n^3)$-depth with linear width [29]. Van Meter and Itoh explore many different approaches for serial QPF, with their lowest achievable depth being $O(n^2 \log n)$ with $O(n^2)$ on NTC [5]. Cleve and Watrous calculate a theoretical minimum circuit depth of $O(\log^3 n)$ and corresponding circuit size of $O(n^3)$ on AC, using an adder which has depth $O(\log n)$ and $O(n)$ size and width. We meet this bound and provide a concrete architectural implementation using an adder with $O(1)$-depth and $O(n)$ size and width.

In the current work, we assume that errors do not affect the storage of qubits during the circuit's operation. An alternate approach is taken by Miquel [30] and Garcia-Mata [31], who both numerically simulate Shor's algorithm for factoring specific numbers to determine its sensitivity to errors. Beckman et al. provide a concrete factoring implementation in ion traps with $O(n^3)$ depth and size and $O(n)$ width [32].

In all the previous works, it is assumed that qubits are expensive (width) and that execution time (depth) is not the limiting constraint. We make the alternative assumption that ancillae are cheap and that fast classical control is available which allows access to all qubits in parallel. Therefore, we optimize circuit depth at the expense of width. We compare our work primarily to Kutin's method [3].

These works also rely on serial QPF which in turn relies on an inverse QFT. On an AC architecture, even when approximating the (inverse) QFT by truncating two-qubit $\pi/2^k$ rotations beyond $k = O(\log n)$, the depth is $O(n \log n)$ to factor an n-bit number. To be implemented fault-tolerantly on a quantum device, rotations in the QFT must then be compiled into a discrete gate basis. This requires at least a $O(\log(1/\epsilon))$ overhead in depth to approximate a rotation with precision ϵ [33, 9]. We would like to avoid the use of a QFT due to its compilation overhead.

There is an alternative, parallel version of phase estimation [34, 9], which we call *parallel QPF* (we refer the reader to Section 13 of [9] for details), which decreases depth in exchange for increased width and additional classical post-processing. This eliminates the need to do an inverse QFT. We develop a nearest-neighbor factoring circuit based on parallel QPF and our proposed 2D quantum arithmetic circuits. We show that it is asymptotically more efficient than the serial QPF method. We compare the circuit resources required by our work with existing serial QPF implementations in Table 3 of Section 8. However, a recent result by [35] allows one to enact a QFT using only Clifford gates and a Toffoli gate in $O(\log^2 n)$ expected depth. This would allow us to greatly improve the constants in our circuit resource upper bounds in Section 7 by combining a QFT with parallel multiplication similar to the approach described in [5, 34].

We also note that recent results by Browne, Kashefi, and Perdrix (BKP) connect the power of measurement-based quantum computing to the quantum circuit model augmented with unbounded fanout [15]. Their model, which we adapt and call CCNTC, uses the classical controller mentioned in Section 2.5. Using results by Høyer and Špalek [36] that unbounded quantum fanout would allow or a constant-depth factoring algorithm, they conclude that a probabilistic polytime classical machine with access to a constant-depth one-way quantum computer would also be able to factor.

4 The Constant-Depth Carry-Save Technique

Our 2D factoring approach rests on the central technique of the constant-depth carry-save adder (CSA) [19], which converts the sum of three numbers a, b, and c, to the sum of two numbers u and v: $a + b + c = u + v$. The explanation of this technique and how it achieves constant depth requires the following definitions.

A *conventional number* x can be represented in n bits as $x = \sum_{i=0}^{n-1} 2^i x_i$, where $x_i \in \{0, 1\}$ denotes the i-th bit of x, which we call an i-bit and has significance 2^i, and the 0-th bit is the low-order bit.[c] Equivalently, x can be represented as a (non-unique) sum of two smaller, $(n-1)$-bit, conventional numbers, u and v. We say $(u+v)$ is a *carry-save encoded*, or CSE, number. The CSE representation of an n-bit conventional number consists of $2n-2$ individual bits where v_0 is always 0 by convention.

Consider a CSA operating on three bits instead of three numbers; then a CSA converts the sum of three i-bits into the sum of an i-bit (the *sum* bit) and an $(i+1)$-bit (the *carry* bit): $a_i + b_i + c_i = u_i + v_{i+1}$. By convention, the bit u_i is the parity of the input bits $(u_i = a_i \oplus b_i \oplus c_i)$ and the bit v_{i+1} is the majority of $\{a_i, b_i, c_i\}$. Figure 5 gives a concrete example, where $(u + v)$ has $2n - 2 = 8$ bits, not counting v_0.

It will also be useful to refer to a subset of the bits in a conventional number using subscripts to indicate a range of indices:

$$x_{(j,k)} \equiv \sum_{i=j}^{k} 2^i x_i \qquad x_{(i)} \equiv x_{(i,i)} = 2^i x_i. \tag{6}$$

Using this notation, the following identity holds:

$$x_{(j,k)} = x_{(j,\ell)} + x_{(\ell+1,k)}, \qquad \text{for all } j \leq \ell < k. \tag{7}$$

We can express the relationship between the bits of x and $(u + v)$ as follows:

$$x = x_{(0,n-1)} \equiv u + v = u_{(0,n-2)} + v_{(1,n-1)}. \tag{8}$$

[c] It will be clear from the context whether we mean an i-bit, which has significance 2^i, or an i-bit number.

$$x = 30 = u + v = 8 + 22 = \left\{ \begin{array}{ccccc} & u_3 & u_2 & u_1 & u_0 \\ v_4 & v_3 & v_2 & v_1 & \\ \hline x_4 & x_3 & x_2 & x_1 & x_0 \end{array} \right\} = \left\{ \begin{array}{ccccc} & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & \\ \hline 1 & 1 & 1 & 1 & 0 \end{array} \right\}$$

Fig. 5. An example of carry-save encoding for the 5-bit conventional number 30.

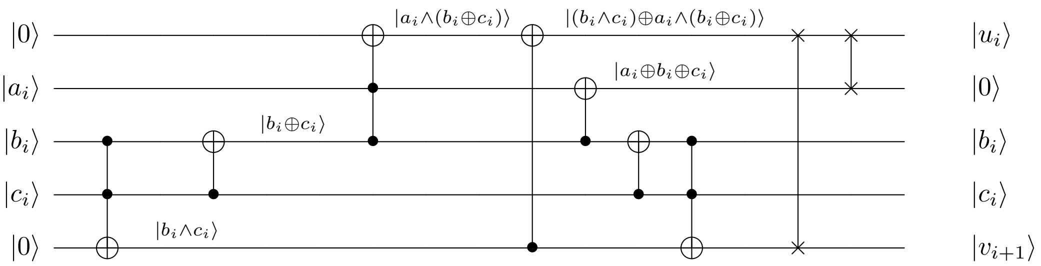

Fig. 6. Carry-save adder circuit for a single bit position i: $a_i + b_i + c_i = u_i + v_{i+1}$.

Finally, we denote arithmetic modulo m with square brackets.

$$x_{(j,k)} \bmod m = x_{(j,k)}[m] \tag{9}$$

Figure 6 gives a circuit description of carry-save addition (CSA) for a single bit position i. The resources for this circuit are given in Table 2, using the resources for the Toffoli gate (in the same table) based on [37]. We note here that a more efficient decomposition for the Toffoli is possible using a distillation approach described in [38].

We must lay out the circuit to satisfy a 2D NTC model. The Toffoli gate decomposition in [37], duplicated in Figure 7, requires two control qubits and a single target qubit to be mutually connected to each other. Given this constraint, and the interaction of the CNOTs in Figure 6, we can rearrange these qubits on a 2D planar grid and obtain the layout shown in Figure 8, which satisfies our 2D NTC model. Qubits $|a\rangle_i$, $|b\rangle_i$, and $|c\rangle_i$ reside at the top of Figure 8, while qubits $|u_i\rangle$ and $|v_{i+1}\rangle$ are initialized to $|0\rangle$. Upon completion of the circuit, qubit $|a_i\rangle$ is in state $|0\rangle$, as seen from the output in Figure 6. Note that this construction uses more gates and one more ancilla than the equivalent quantum full adder circuit in Figure 5 of [19], however this is necessary in order to meet our architectural constraints and does not

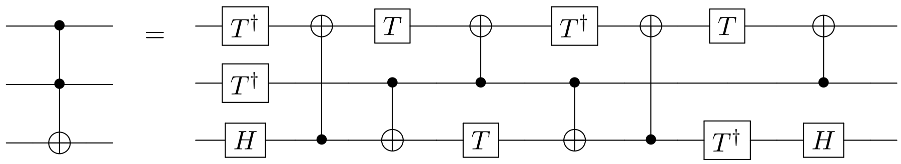

Fig. 7. The depth-efficient Toffoli gate decomposition from [37].

Circuit Name	Depth	Size	Width
Toffoli gate from [37] and Figure 7	8	15	3
Single-bit 3-to-2 adder from Figure 6	33	55	5

Table 2. Circuit resources for Toffoli and single-bit addition.

change the asymptotic results. Also in Figure 8 is a variation called a 2-2 adder, which simply re-encodes two i-bits into an i-bit and an $(i+1)$-bit. This will be useful in the next section.

At the level of numbers, the sum of three n-bit numbers can be converted into the sum of two n-bit numbers by applying a *CSA layer* of n parallel, single-bit CSA circuits (Fig. 6). Since each CSA operates in constant depth, the entire layer also operates in constant depth, and we have achieved (non-modular) addition. Each single addition of three n-bit numbers requires $O(n)$ circuit width.

5 Quantum Modular Addition

To perform addition of two numbers a and b modulo m, we consider the variant problem of modular addition of three numbers to two numbers: Given three n-bit input numbers a, b, and c, and an n-bit modulus m, compute $(u+v) = (a+b+c)[m]$, where $(u+v)$ is a CSE number.

In this section, we provide an alternate, pedagogical explanation of Gossett's modular reduction [19]. Later, we contribute a mapping of this adder to a 2D architecture, using unbounded fanout to maintain constant depth for adding back modular residues. This last step is absent in Gossett's original approach.

To start, we will demonstrate the basic method of modular addition and reduction on an n-bit conventional number. In general, adding two n-bit conventional numbers will produce an overflow bit of significance 2^n, which we can truncate as long as we add back its modular residue $2^n \bmod m$. How can we guarantee that we won't generate another overflow bit by adding back the modular residue? It turns out we can accomplish this by allowing a slightly larger input and output number ($n+1$ bits in this case), truncating multiple overflow bits, and adding back their modular residues.

For an $(n+1)$-bit conventional number x, we truncate its high-order bits x_n and x_{n-1} and add back their modular residue $x_{(n-1,n)}[m]$:

$$\begin{aligned} x \bmod m &= x_{(0,n)}[m] \\ &= x_{(0,n-2)} + x_{(n-1,n)}[m]. \end{aligned} \tag{10}$$

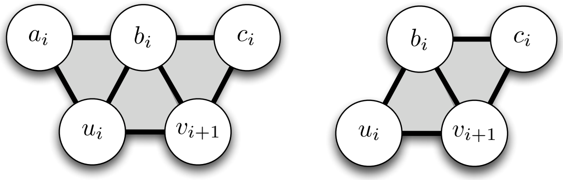

Fig. 8. The carry-save adder (CSA), or 3-2 adder, and carry-save 2-2 adder.

Since both the truncated number $x_{(0,n-2)}$ and the modular residue are n-bit numbers, their sum is an $(n+1)$-bit number as desired, equivalent to $x[m]$.

Now we must do the same modular reduction on a CSE number $(u+v)$, which in this case represents an $(n+2)$-bit conventional number and has $2n+3$ bits. First, we truncate the three high-order bits (v_n, u_{n-1}, v_{n-1}) of $(u+v)$, yielding an n-bit conventional number with a CSE representation of $2n$ bits: $\{u_0, u_1, \ldots, u_{n-1}\} \cup \{v_1, v_2, \ldots, v_{n-1}\}$. Then we add back the three modular residues $(v_{(n+1)}[m], u_{(n)}[m], v_{(n)}[m])$, and we are guaranteed not to generate additional overflow bits (of significance 2^n or higher). This equivalence is shown in Eq 11.

$$\begin{aligned}(u+v)[m] &= \left(u_{(0,n+1)} + v_{(1,n+2)}\right)[m] \\ &= u_{(0,n)} + v_{(1,n)} + \\ &\quad u_{(n+1)}[m] + v_{(n+1)}[m] + \\ &\quad v_{(n+2)}[m] \end{aligned} \qquad (11)$$

[Modular Reduction in Constant Depth] The modular addition of three n-bit numbers to two n-bit numbers can be accomplished in constant depth with $O(n)$ width in 2D CCNTC.

Proof: Our goal is to show how to perform modular addition while keeping our numbers of a fixed size by treating overflow bits correctly. We map the proof of [19] to 2D CCNTC and show that we meet our required depth and width. First, we enlarge our registers to allow the addition of $(n+2)$-bit numbers, while keeping our modulus of size n bits. (In Gossett's original approach, he takes the equivalent step of restricting the modulus to be of size $(n-2)$ bits.) We accomplish the modular addition by first performing a layer of non-modular addition, truncating the three high-order overflow bits, and then adding back modular residues controlled on these bits in three successive layers, where we are guaranteed that no additional overflow bits are generated in each layer. This is illustrated for a 3-bit modulus and 5-bit registers in Figure 9.

We use the following notation. The non-modular sum of the first layer is u and v. The CSE output of the first modular reduction layer is u' and v', and the modular residue is written as $c^{v_{n+1}}$ to mean the precomputed value $2^{n+1} \bmod m$ controlled on v_{n+1}. The CSE output of the second modular reduction layer is u'' and v'', and the modular residue is written as $c^{u_{n+1}}$ to mean the precomputed value $2^{n+1} \bmod m$ controlled on u_{n+1}. The CSE output of the third and final modular reduction layer is u''' and v''', and the modular residue is written as $c^{v_{n+2}}$ to mean the precomputed value $2^{n+2} \bmod m$ controlled on v_{n+2}.

We show that no layer generates an overflow $(n+2)$-bit, namely in the v component of any CSE output. (The u component will never exceed the size of the input numbers.) First, we know that no v'_{n+2} bit is generated after the first modular reduction layer, because we have truncated away all $(n+1)$-bits. Second, we know that no v''_{n+2} bit is generated because we only have one $(n+1)$-bit to add, v'_{n+1}. Finally, we need to show that $v'''_{n+2} = 0$ in the third modular reduction layer.

Since $u'_{(n)} + v'_{(n+1)} = u_{(n)} + v_{(n)} \leq 2^{n+1}$, the bits u'_n and v'_{n+1} cannot both be 1. But $u''_{n+1} = v'_{n+1}$ and $v''_{n+1} = u'_n \wedge v'_n$, so u''_{n+1} and v''_{n+1} cannot both be 1, and hence $v'''_{n+2} = 0$. Everywhere we use the fact that the modular residues are restricted to n bits. Therefore,

$$
\begin{array}{cccccclr}
 & a_4 & a_3 & a_2 & a_1 & a_0 & \text{5-bit input number } a & \\
 & b_4 & b_3 & b_2 & b_1 & b_0 & \text{5-bit input number } b & \\
 & c_4 & c_3 & c_2 & c_1 & c_0 & \text{5-bit input number } c & \text{[Layer 1]} \\
\hline
 & u_4 & u_3 & u_2 & u_1 & u_0 & \text{truncate } u_4 & \\
v_5 & v_4 & v_3 & v_2 & v_1 & & \text{truncate } v_4, v_5 & \\
 & & & c_2^{v_4} & c_1^{v_4} & c_0^{v_4} & \text{add back } 2^4 \bmod m \text{ controlled on } v_4 & \text{[Layer 2]} \\
\hline
 & & u_3' & u_2' & u_1' & u_0' & & \\
 & v_4' & v_3' & v_2' & v_1' & & & \\
 & & & c_2^{u_4} & c_1^{u_4} & c_0^{u_4} & \text{add back } 2^4 \bmod m \text{ controlled on } u_4 & \text{[Layer 3]} \\
\hline
 & u_4'' & u_3'' & u_2'' & u_1'' & u_0'' & \text{the bit } u_4'' \text{ is the same as } v_4' & \\
 & v_4'' & v_3'' & v_2'' & v_1'' & & & \\
 & & & c_2^{v_5} & c_1^{v_5} & c_0^{v_5} & \text{add back } 2^5 \bmod m \text{ controlled on } v_5 & \text{[Layer 4]} \\
\hline
 & u_4''' & u_3''' & u_2''' & u_1''' & u_0''' & \text{Final CSE output with 5 bits} & \\
 & v_4''' & v_3''' & v_2''' & v_1''' & & \text{Final CSE output with 5 bits} & \\
\end{array}
$$

Fig. 9. A schematic proof of Gossett's constant-depth modular reduction for $n = 3$.

the modular sum is computed as the sum of two $(n + 2)$-bit numbers with no overflows in constant-depth. □

As a side note, we can perform modular reduction in one layer instead of three by decoding the three overflow bits into one of seven different modular residues. This can also be done in constant depth, and in this case we only need to enlarge all our registers to $(n + 1)$ bits instead of $(n + 2)$ as in the proof above. We omit the proof for brevity.

In the following two subsections, we give a concrete example to illustrate the modular addition circuit as well as a numerical upper bound for the general circuit resources.

5.1 A Concrete Example of Modular Addition

A 2D CCNTC circuit for modular addition of 5-bit numbers using four layers of parallel CSA's is shown graphically in Figure 10 which corresponds directly to the schematic proof in Figure 9. Note that in Figure 10, the least significant qubits are on the left, and in Figure 9, the least significant qubits are on the right. Figure 10 also represents the approximate physical layout of the qubits as they would look if this circuit were to be fabricated. Here, we convert the sum of three 5-bit integers into the modular sum of two 5-bit integers, with a 3-bit modulus m. In the first layer, we perform 4 CSA's in parallel on the input numbers (a, b, c) and produce the output numbers (u, v).

As described above, we truncate the three high-order bits during the initial CSA round (bits u_4, v_4, v_5) to retain a 4-bit number. Each of these bits serves as a control for adding its modular residue to a running total. We can classically precompute $2^4[m]$ for the two additions controlled on u_4 and v_4 and $2^5[m]$ for the addition controlled on v_5.

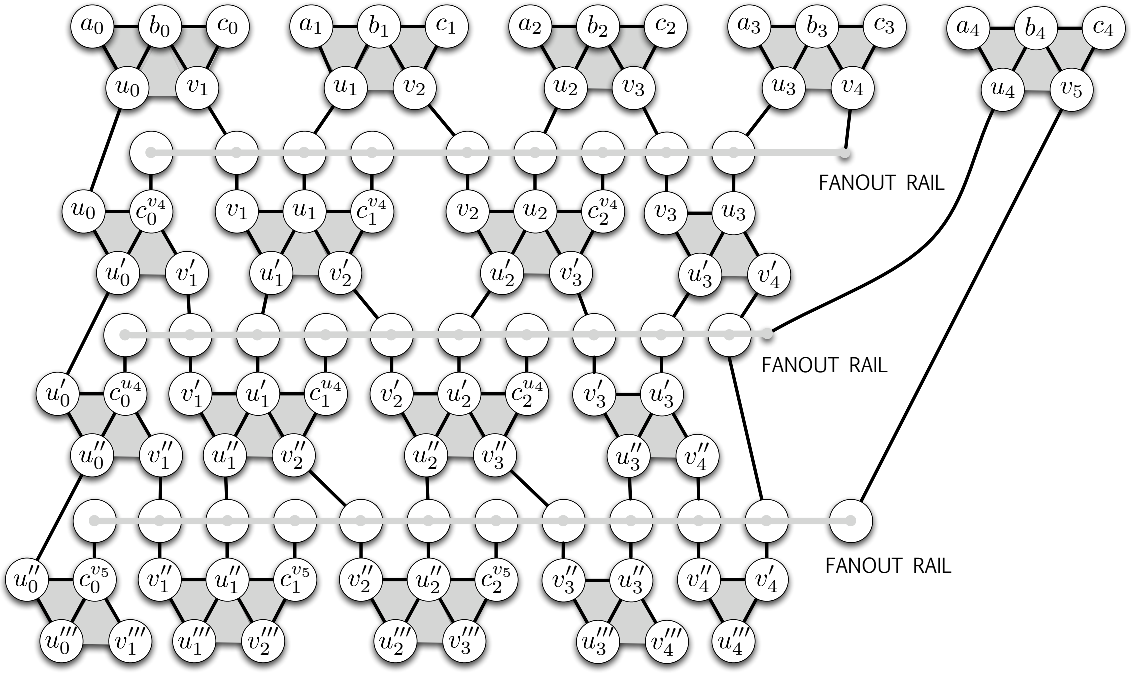
a_0 b_0 c_0 a_1 b_1 c_1 a_2 b_2 c_2 a_3 b_3 c_3 a_4 b_4 c_4
u_0 v_1 u_1 v_2 u_2 v_3 u_3 v_4 u_4 v_5
FANOUT RAIL
u_0 $c_0^{v_4}$ v_1 u_1 $c_1^{v_4}$ v_2 u_2 $c_2^{v_4}$ v_3 u_3
u'_0 v'_1 u'_1 v'_2 u'_2 v'_3 u'_3 v'_4
FANOUT RAIL
u'_0 $c_0^{u_4}$ v'_1 u'_1 $c_1^{u_4}$ v'_2 u'_2 $c_2^{u_4}$ v'_3 u'_3
u''_0 v''_1 u''_1 v''_2 u''_2 v''_3 u''_3 v''_4
FANOUT RAIL
u''_0 $c_0^{v_5}$ v''_1 u''_1 $c_1^{v_5}$ v''_2 u''_2 $c_2^{v_5}$ v''_3 u''_3 v''_4 v'_4
u'''_0 v'''_1 u'''_1 v'''_2 u'''_2 v'''_3 u'''_3 v'''_4 u'''_4

Fig. 10. Addition and three rounds of modular reduction for a 3-bit modulus.

In Layer 2, we use a constant-depth fanout rail (see Figure 4) to distribute the control bit v_4 to its modular residue, which we denote as $|c^{v_4}\rangle \equiv \left|2^4[m] \cdot v_4\right\rangle$. c^{v_4} has n bits, which we add to the CSE results of layer 1. The results u_i and v_{i+1} are teleported into layer 3. The exception is v'_4 which is teleported into layer 4, since there are no other 4-bits to which it can be added. Wherever there are only two bits of the same significance, we use the 2-2 adder from Section 4.

Layer 3 operates similarly to layer 2, except that the modular residue is controlled on u_4: $|c^{u_4}\rangle \equiv \left|2^4[m] \cdot u_4\right\rangle$. The qubit c^{u_4} has 3 bits, which we add to the CSE results of layer 2, where u'_i and v'_{i+1} are teleported forward into layer 4.

Layer 4 is similar to layers 2 and 3, with the modular residue controlled on v_5: $|c^{v_5}\rangle \equiv \left|2^5[m] \cdot v_5\right\rangle$. The qubit c^{v_5} has 3 bits, which we add to the CSE results of layer 3. There is no overflow bit v'''_5, and no carry bit from v''_4 and v'_4 as argued in Lemma 1. The final modular sum $(a + b + c)[m]$ is $u''' + v'''$.

The general circuit for adding three n-qubit quantum integers to two n-qubit quantum integers is called a *CSA tile*. Each CSA tile in our architecture corresponds to its own module, and it will be represented by the symbol in Figure 11 for the rest of this paper. We call this an n-bit modular adder, even though it accepts $(n + 2)$-bit inputs, because the size of the modulus is still n bits.

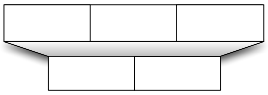

Fig. 11. Symbol for an n-bit 3-to-2 modular adder, also called a CSA tile.

5.2 Quantum Circuit Resources for Modular Addition

We now calculate numerical upper bounds for the circuit resources of the n-bit 3-to-2 modular adder described in the previous section. There are four layers of non-modular n'-bit 3-to-2 adders, which consists of n' parallel single-bit adders whose resources are detailed in Table 2. For factoring an n-bit modulus, we have $n' = n + 2$ in the first and fourth layers and $n' = n + 1$ in the second and third layers.

After each of the first three layers, we must move the output qubits across the fanout rail to be the inputs of the next layer. We use two swap gates, which have a depth and size of 6 CNOTs each, since the depth of teleportation is only more efficient for moving more than two qubits. The control bit for each modular residue needs to be teleported 0, 4, and 7 qubits respectively according to the diagram in Figure 10, before being fanned out n times along the fanout rails, where the fanned out copies will end up in the correct position to be added as inputs.

The resources for the n-bit 3-to-2 modular adder depicted in Figure 10 is more complicated due to the un-fanout procedure. The formulae below reflect the resources needed for both computing the output in the forward direction (creating an entangled fanned-out state) and also uncomputing ancillae in the backward direction (disentangling fanned-out copies).

The circuit depth is:

$$356 + 8\log_2(2n + 4). \tag{12}$$

The circuit size is:

$$33n\log_2 n + 40\log^2 n + 575n + 752. \tag{13}$$

The circuit width is:

$$33n + 47. \tag{14}$$

6 Quantum Modular Multiplication

We can build upon our carry-save adder to implement quantum modular multiplication in logarithmic depth. We start with a completely classical problem to illustrate the principle of multiplication by repeated addition. Then we consider modular multiplication of two quantum integers in a serial and a parallel fashion in Section 6.1. Both of these problems use as a subroutine the generic problem of *modular multiple addition* which we define and solve in Section 6.3.

First we consider a completely classical problem: given three n-bit classical numbers a, b, and m, compute $c = ab \bmod m$, where c is allowed to be in CSE.

We only have to add shifted multiples of a to itself, "controlled" on the bits of b. There are n shifted multiples of a, let's call them $z^{(i)}$, one for every bit of b: $z^{(i)} = 2^i ab_i \bmod m$. We can parallelize the addition of n numbers in a logarithmic depth binary tree to get a total depth of $O(\log n)$.

6.1 Modular Multiplication of Two Quantum Integers

We now consider the problem of multiplying a classical number controlled on a quantum bit with a *quantum integer*,[d] which is a quantum superposition of classical numbers:

> Given an n-qubit quantum integer $|x\rangle$, a control qubit $|p\rangle$, and two n-bit classical numbers a and m, compute $|c\rangle = |xa[m]\rangle$, where c is allowed to be in CSE.

This problem occurs naturally in modular exponentiation (described in the next section) and can be considered *serial multiplication*, in that t quantum integers are multiplied in series to a single quantum register. This is the approach used in serial QPF as mentioned in Section 3.

We first create n quantum integers $\left|z^{(i)}\right\rangle$, which are shifted multiples of the classical number a controlled on the bits of x: $\left|z^{(i)}\right\rangle \equiv \left|2^i a[m] \cdot x_i\right\rangle$. These are typically called *partial products* in a classical multiplier. How do we create these numbers, and what is the depth of the procedure? First, note that $\left|2^i a[m]\right\rangle$ is a classical number, so we can precompute them classically and prepare them in parallel using single-qubit operations on n registers, each consisting of n ancillae qubits. Each n-qubit register will hold a future $\left|z^{(i)}\right\rangle$ value. We then fan out each of the n bits of x, n times each, using an unbounded fanout operation so that n copies of each bit $|x_i\rangle$ are next to register $\left|z^{(i)}\right\rangle$. This takes a total of $O(n^2)$ parallel CNOT operations. We then entangle each $\left|z^{(i)}\right\rangle$ with the corresponding x_i. After this, we interleave these numbers into groups of three using constant-depth teleportation. This reduces to the task of modular multiple addition in order to add these numbers down to a single (CSE) number modulo m, which is described in Section 6.3.

Finally, we tackle the most interesting problem:

> Given two n-qubit quantum integers $|x\rangle$ and $|y\rangle$ and a n-bit classical number m, compute $|c\rangle = |xy \bmod m\rangle$, where $|c\rangle$ is allowed to be in CSE.

This can be considered *parallel multiplication* and is responsible for our logarithmic speedup in modular exponentiation and parallel QPF.

Instead of creating n quantum integers $\left|z^{(i)}\right\rangle$, we must create up to n^2 numbers $\left|z^{i,j}\right\rangle$ for all possible pairs of quantum bits x_i and y_j, $i, j \in \{0, \ldots, n-1\}$: $\left|z^{i,j}\right\rangle \equiv \left|2^i 2^j [m] \cdot x_i \cdot y_j\right\rangle$. We create these numbers using a similar procedure to the previous problem. Adding n^2 quantum integers of n qubits each takes depth $O(\log(n^2))$, which is still $O(\log n)$. Creating $n^2 \times n$-bit quantum integers takes width $O(n^3)$. Numerical constants are given for these resource estimates in Section 6.4 for the entire modular multiplier.

Here is an outline of our modular multiplier construction, combining the two halves of partial product creation (Section 6.2) and modular multiple addition (Section 6.3).

[d] In this paper, quantum integers often result by entangling a classical number in one register with a quantum control bit.

1. Initially, the inputs consist of the CSE quantum integers x and y, each with $2n+3$ bits, sitting on adjacent edges of a square lattice that has a length of $3(2n+3)$ qubits.
2. For each of $\lceil \log_2(2n+3) \rceil$ rounds:
   (a) Of the existing $\{x_i\}$ and $\{y_j\}$ bits, apply a CNOT to create an entangled copy in an adjacent qubit.
   (b) Teleport this new copy halfway between its current location and the new copy.
   (c) At every site where an $|x_i\rangle$ and an $|y_j\rangle$ meet, apply a Toffoli gate to create $|x_i \cdot y_j\rangle$.
   (d) Teleport $|x_i \cdot y_j\rangle$ to the correct z-site module.
3. Within each z-site module, fanout $|x_i \cdot y_j\rangle$ up to n times, corresponding to each 1 in the modular residue $2^i 2^j \bmod m$, to create the n-qubit quantum integer $\left|z^{(i,j)}\right\rangle$.
4. For each triplet of z-site modules, teleport the quantum integers $\left|z^{(i,j)}\right\rangle$ to a CSA tile module, interleaving the three numbers so that bits of the same significance are adjacent. This concludes partial product creation (Section 6.2).
5. Perform modular multiple addition (described in Section 6.3) on t' n-qubit quantum integers down to 2 n-qubit quantum integers (one CSE number).
6. Uncompute all the previous steps to restore ancillae to $|0\rangle$.

6.2 Partial Product Creation

This subroutine describes the procedure of creating $t = O(n^2)$ partial products of the CSE quantum integers x and y, each with $2n+3$ bits each. We will now discuss only the case of parallel multiplication. Although we will not provide an explicit circuit for this subroutine, we will outline our particular construction and give a numerical upper bound on the resources required.

First, we need to generate the product bits $|x_i \cdot y_j\rangle$ for all possible $(2n+3)^2$ pairs of $|x_i\rangle$ and $|y_j\rangle$. A particular product bit $|x_i \cdot y_j\rangle$ controls a particular classical number, the n-bit modular residue $2^i 2^j[m]$, to form the partial product $\left|z^{(i,j)}\right\rangle$ defined in the previous section. However, some of these partial products consist of only a single qubit, if $2^i 2^j < 2^n$, which is the minimum value for an n-bit modulus m. There are at least $2n^2 - 2n + 1$ such single-bit partial products, which can be grouped into at most $(2n+3) \times n$-bit numbers. Of the $(2n+3)^2$ possible partial products, this leaves the number of remaining n-bit partial products as at most $2n^2 + 14n + 8$. Therefore we have a final maximum number of n-bit partial products, which we will simply refer to as t' from now on.

$$t' = 2n^2 + 16n + 11 \tag{15}$$

The creation of the product bits $|x_i \cdot y_j\rangle$ occurs on a square lattice of $(3(2n+3))^2$, with the numbers $|x_i\rangle$ and $|y_j\rangle$ on adjacent edges. The size of the lattice allows the $|x_i\rangle$ and $|y_j\rangle$ bits move past each other. The $|x_i\rangle$ bits are teleported along an axis that is perpendicular to the teleportation axis for the $|y_j\rangle$ bits, and vice versa. Product bit creation, and this square lattice, comprise a single module. In several rounds, these bits are copied via a CNOT and teleported to the middle of a recursively halved interval of the grid. The copied bits $|x_i\rangle$ and

$|y_j\rangle$ first form 1 line, then 3 lines, then 7 lines, and so forth, intersecting at 1 site, then 9 sites, then 49 sites, and so forth. There are $\lceil \log_2(2n+3) \rceil$ such rounds.

At each intersection, a Toffoli gate is used to create $|x_i \cdot y_j\rangle$ from the given $|x_i\rangle$ and $|y_j\rangle$. These product bits are then teleported away from this qubit, out of this product bit module, to different modules where the $\left|z^{(i,j)}\right\rangle$ numbers are later generated, called z-sites. There are t' z-site modules which each contain an n-qubit quantum integer. Any round of partial product generation will produce at most as many product bits $x_i \cdot y_j$ as in the last round, which is half the total number of $(2n+3)^2$.

We now present the resources for partial product creation, the first half of a modular multiplier, including the reverse computation.

The circuit depth is $O(\log n)$:

$$D_{PPC} = 40 \log_2 10n. \tag{16}$$

The module depth is $O(1)$:

$$\overline{D}_{PPC} = 8. \tag{17}$$

The circuit size is $O(n^2 \log^2 n)$:

$$\begin{aligned} S_{PPC} \quad = \quad & (12n^2 + 84n + 48) \log_2^2(10n) + & (18)\\ & (8n^3 + 64n^2 + 94n + 31) \log_2(10n) + & (19)\\ & (78n^3 + 597n^2 + 419n + 133). & (20) \end{aligned}$$

The module size is $O(n^2)$:

$$\overline{S}_{PPC} = 6n^2 + 26n + 19. \tag{21}$$

The circuit width is $O(n^3)$:

$$W_{PPC} = 6n^3 + 48n^2 - 8n + 1. \tag{22}$$

The module width is $O(n^2)$:

$$\overline{W}_{PPC} = 2n^2 + 14n + 9. \tag{23}$$

6.3 Modular Multiple Addition

As a subroutine to modular multiplication, we define the operation of repeatedly adding multiple numbers down to a single CSE number, called *modular multiple addition*.

The modular multiple addition circuit generically adds down $t' \times n$-bit conventional numbers to an n-bit CSE number:

$$z^{(1)} + z^{(2)} + \ldots z^{(t')} \equiv (u + v)[m]. \tag{24}$$

It does not matter how the t' numbers are generated, as long as they are divided into groups of three and have their bits interleaved to be the inputs of a CSA tile. From the previous section, serial multiplication results in $t' \leq n$ and parallel multiplication results in $t' \leq n^2$. Each CSA tile is contained in its own module. These modules are arranged in layers within a logarithmic depth binary tree, where the first layer contains $\lceil t'/3 \rceil$ modules. A modular addition occurs in all the modules of the first layer in parallel. The outputs from this first layer are then teleported to be the inputs of the next layer of modules, which have at most two-thirds as many modules. This continues until the tree terminates in a single module, whose output is a CSE number $u + v$ which represents the modular product of all the original t' numbers. The resulting height of the tree is $(\lceil \log_{3/2}(t'/3) \rceil + 1)$ modules.

As the parallel modular additions proceed by layers, all previous layers must be maintained in a coherent state, since the modular addition leaves garbage bits behind. Only at the end of modular multiple addition, after the final answer $u + v$ is obtained, can all the previous layers be uncomputed in reverse to free up their ancillae.

These steps are best illustrated with a concrete example in Figure 12. The module for each CSA tile is represented by the symbol from Figure 11. The arrows indicate the teleportation of output numbers from the source tile to be input numbers into a destination tile.

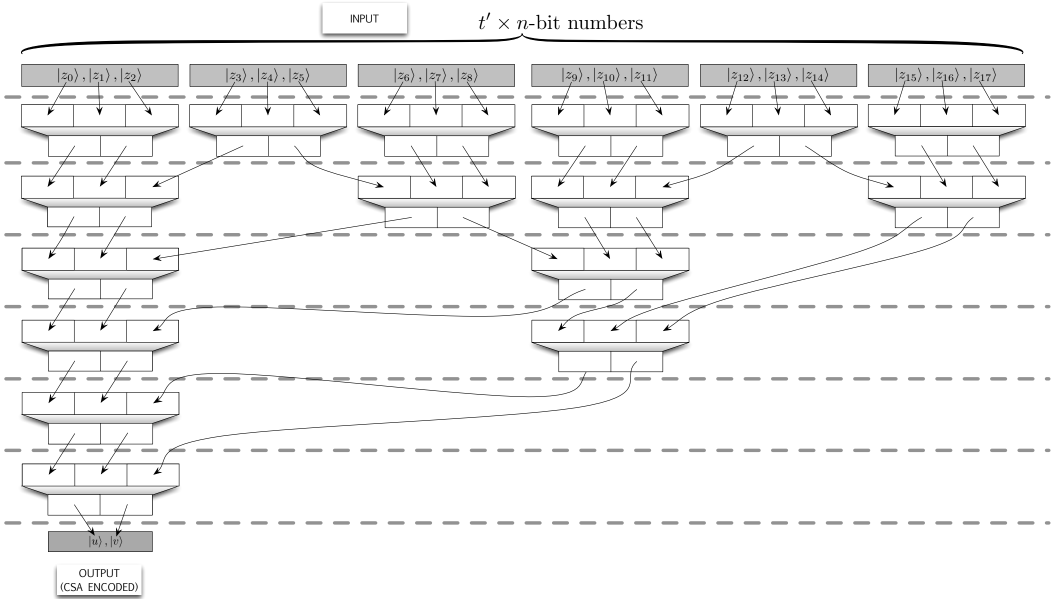
INPUT
$t' \times n$-bit numbers
$|z_0\rangle, |z_1\rangle, |z_2\rangle$
$|z_3\rangle, |z_4\rangle, |z_5\rangle$
$|z_6\rangle, |z_7\rangle, |z_8\rangle$
$|z_9\rangle, |z_{10}\rangle, |z_{11}\rangle$
$|z_{12}\rangle, |z_{13}\rangle, |z_{14}\rangle$
$|z_{15}\rangle, |z_{16}\rangle, |z_{17}\rangle$
$|u\rangle, |v\rangle$
OUTPUT
(CSA ENCODED)

Fig. 12. Modular multiple addition of quantum integers on a CSA tile architecture for $t' = 18$ in a logarithmic-depth tree with height $(\lceil \log_{\frac{3}{2}}(t'/3) \rceil + 1) = 6$. Arrows represent teleportation in between modules.

Now we can analyze the circuit resources for multiplying n-bit quantum integers, which requires $(t' - 2)$ modular additions, for t' from Equation 15. The circuit width is the sum of the $O(n^3)$ ancillae needed for partial product creation and the ancillae required for $O(n^2)$ modular additions. Each modular addition has width $O(n)$ and depth $O(1)$ from the previous section. There are $\lceil \log_{3/2}(n^2/3) \rceil + 1$ timesteps of modular addition. Therefore the entire modular multiplier circuit has depth $O(\log n)$ and width $O(n^3)$.

6.4 Modular Multiplier Resources

.

The circuit depth of the entire modular multiplier is $O(\log^3 n)$:

$$D_{MM} = 3.4 \log_2^3 n + 1295 \log_2 n + 6911. \tag{25}$$

The module depth is $O(\log n)$:

$$\overline{D}_{MM} = 2 \log_2 n + 11. \tag{26}$$

The circuit size is $O(n^3 \log n)$:

$$\begin{aligned} S_{MM} = \quad & (12n^2 + 644n + 288) \log_2^2(10n) + & (27)\\ & (74n^3 + 606n^2 + 292n + 41) \log_2(10n) + & (28)\\ & (1228n^3 + 10151n^2 + 14397n + 4645). & (29) \end{aligned}$$

The module size is $O(n^3)$:

$$\overline{S}_{MM} = 15n^3 + 127n^2 + 178n + 50. \tag{30}$$

The circuit width is $O(n^3)$:

$$W_{MM} = 66n^3 + 558n^2 + 870n + 290. \tag{31}$$

The module width is $O(n^2)$:

$$\overline{W}_{MM} = 4n^2 + 28n + 15. \tag{32}$$

7 Quantum Modular Exponentiation

We now extend our arithmetic to modular exponentiation, which is repeated modular multiplication controlled on qubits supplied by a phase estimation procedure. If we wish to multiply an n-qubit quantum input number $|x\rangle$ by t classical numbers $a^{(j)}$, we can multiply them in series. This requires depth $O(t \log n)$ in modular multiplication operations.

Now consider the same procedure, but this time each classical number $a^{(j)}$ is controlled on a quantum bit p_j. This is a special case of multiplying by t quantum integers in series, since a classical number entangled with a quantum integer is also quantum. It takes the same depth $O(t \log n)$ as the previous case.

Finally, we consider multiplying t quantum integers $\{x^{(1)}, x^{(2)}, \ldots, x^{(t-1)}, x^{(t)}\}$ in a parallel, logarithmic-depth binary tree. This is shown in Figure 13, where arrows indicate multiplication. The tree has depth $\log_2(t)$ in modular multiplier operations. Furthermore, each modular multiplier has depth $O(\log^2(n))$ and width $O(n^3)$ for n-qubit numbers. Therefore, the overall depth of this parallel modular exponentiation structure is $O(\log(t) \log^2(n))$ with width $O(tn^3)$. In phase estimation for QPF, it is sufficient to take $t = O(n)$ [13, 9]. Therefore our total depth is $O(\log^3(n))$ and our total width is $O(n^4)$, as desired. At this point,

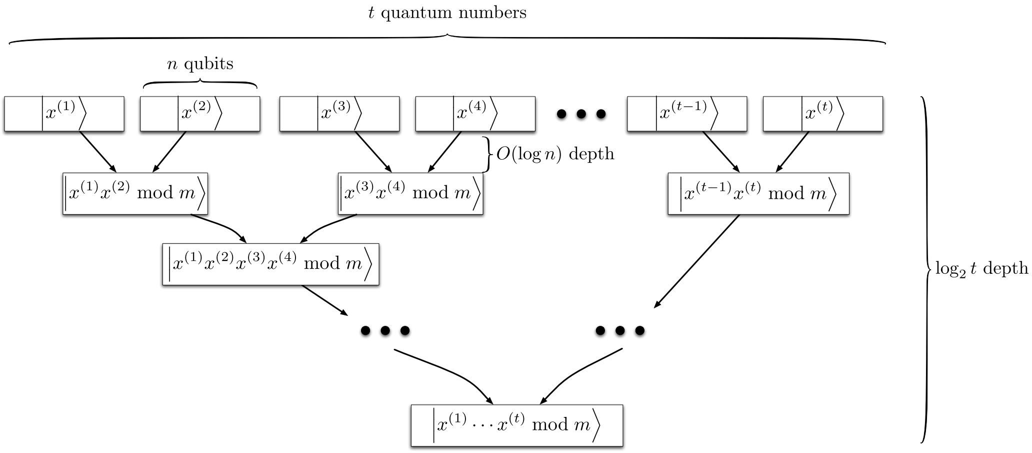
t quantum numbers
n qubits
$|x^{(1)}\rangle$
$|x^{(2)}\rangle$
$|x^{(3)}\rangle$
$|x^{(4)}\rangle$
$|x^{(t-1)}\rangle$
$|x^{(t)}\rangle$
$O(\log n)$ depth
$|x^{(1)}x^{(2)} \bmod m\rangle$
$|x^{(3)}x^{(4)} \bmod m\rangle$
$|x^{(t-1)}x^{(t)} \bmod m\rangle$
$|x^{(1)}x^{(2)}x^{(3)}x^{(4)} \bmod m\rangle$
$|x^{(1)}\cdots x^{(t)} \bmod m\rangle$
$\log_2 t$ depth

Fig. 13. Parallel modular exponentiation: multiplying t quantum integers in a $O(\log(t)\log(n))$-depth binary tree. Arrows indicate modular multiplication.

combined with the parallel phase estimation procedure of [9], we have a complete factoring implementation in our 2D nearest-neighbor architecture in polylogarithmic depth.

We will now calculate numerical constants to upper bound circuit resources.

According to the Kitaev-Shen-Vyalyi parallelized phase estimation procedure [9], for a constant success probability of 3/4, it is sufficient to multiply together $t' = 2867n$ quantum integers, controlled on the qubits $|p_j\rangle$, in parallel.

In Section 7.1, we describe the last step of modular exponentiation in CSE. In Section 7.2, we state the final circuit resources for the entire modular exponentiation circuit, and therefore, our quantum period-finding procedure.

7.1 Converting Back to a Unique Conventional Number

The final product of all t quantum integers is in CSE which is not unique. As stated in Gossett's original paper [19], this must be converted back to a conventional number using, for example, the quantum carry-lookahead adder (QCLA) from [39]. We can convert this to a nearest-neighbor architecture by using the qubit reordering construction of [8]. We now compute the resources needed for this last step.

To add two $(n+2)$-bit numbers in a QCLA, we have a circuit width of $k = (4(n+2) - 2\log_2 n - 1)$. The depth is at most $4\log_2 n + 2$ gates, and some of them act on qubits that are not nearest-neighbors. Therefore, we add in between each gate a reordering circuit that takes k^2 (reusable) ancillae qubits and uses two rounds of constant-depth teleportation to rearrange the qubits into a new order where all the gates are nearest-neighbor. Adding in the teleportation circuit resources from Table 1, we can calculate the following resources.

The circuit depth is $O(\log n)$:

$$56 \log_2 n + 28. \tag{33}$$

The circuit size is $O(n^2 \log n)$:

$$\begin{aligned} 96 \log_2^3 n & - & (384n + 624) \log_2^2 n \\ & + & (384n^2 + 1152n + 840) \log_2 n \\ & + & (192n^2 + 672n + 588). \end{aligned} \tag{34}$$

The circuit width is $O(n^2)$:

$$4 \log_2^2 n - (16n + 30) \log_2 n + 16n^2 + 60n + 56. \tag{35}$$

7.2 Circuit Resources for Modular Exponentiator

This leads to the following circuit resource upper bounds for a modular exponentiator.

The circuit depth is $O(\log^3 n)$:

$$D_{ME} = 3.4 \log_2^3 n + 1337 \log_2^2 n + 23070 \log_2 n + 86279. \tag{36}$$

The module depth is $O(\log n)$:

$$\overline{D}_{ME} = 3 \log_2 n + 24 \tag{37}$$

The circuit size is $O(n^4 \log n)$:

$$\begin{aligned} S_{ME} \quad = \quad & 96 \log_2^3 n + \\ & (34400n^3 + 1846079n^2 + 824937n - 288) \log_2^2 10n + \\ & (212129n^4 + 1737086n^3 + 836442n^2 + 117239n - 41) \log_2 10n + \\ & 3520185n^4 + 29097629n^3 + 41260290n^2 + 13300960n - 4645. \end{aligned} \tag{38}$$

The module size is $O(n^2)$:

$$\overline{S}_{ME} = 5749n^2 + 8725n + 175. \tag{39}$$

The circuit width is $O(n^4)$:

$$W_{ME} = 94598n^4 + 799749n^3 + 1246692n^2 + 415222n - 145. \tag{40}$$

The module width is $O(n)$:

$$\overline{W}_{ME} = 1434n. \tag{41}$$

8 Asymptotic Results

The asymptotic resources required for our approach, as well as the resources for other nearest-neighbor approaches, are listed in Table 3, where we assume some fixed constant error probability for each round of QPF. Not all resources are provided directly by the referenced source.

Resources in square brackets are inferred using Equation 2. These upper bounds are correct, but may not be tight with the upper bounds calculated by their respective authors. In particular, a more detailed analysis could give a better upper bound for circuit size than the depth-width product. Also note that the work by Beckman et al. [32] is unique in that it uses efficient multi-qubit gates inherent to linear ion trap technology which at first seem to be more powerful than 1D NTC. However, use of these gates does not result in an asymptotic improvement over 1D NTC.

We achieve an exponential improvement in nearest-neighbor circuit depth (from quadratic to polylogarithmic) with our approach at the cost of a polynomial increase in circuit size and width. Similar depth improvements at the cost of width increases can be achieved using the modular multipliers of other factoring implementations by arranging them in a parallel modular exponentiator. Our approach is the first implementation for factoring on 2D NTC, augmented with a classical controller and parallel, communicating modules (2D CCNTCM).

Implementation	Architecture	Depth	Size	Width
Vedral, et al. [27]	AC	$[O(n^3)]$	$O(n^3)$	$O(n)$
Gossett [19]	AC	$O(n \log n)$	$[O(n^3 \log n)]$	$O(n^2)$
Beauregard [2]	AC	$O(n^3)$	$O(n^3 \log n)$	$O(n)$
Zalka [28]	AC	$O(n^2)$	$[O(n^3)]$	$O(n)$
Takahashi & Kunihiro [29]	AC	$O(n^3)$	$O(n^3 \log n)$	$O(n)$
Cleve & Watrous [34]	AC	$O(\log^3 n)$	$O(n^3)$	$[O(n^3/\log^3 n)]$
Beckman et al. [32]	Ion trap	$O(n^3)$	$O(n^3)$	$O(n)$
Fowler, et al. [12]	1D NTC	$O(n^3)$	$O(n^4)$	$O(n)$
Van Meter & Itoh [4]	1D NTC	$O(n^2 \log n)$	$[O(n^4 \log n)]$	$O(n^2)$
Kutin [3]	1D NTC	$O(n^2)$	$O(n^3)$	$O(n)$
Current Work	2D CCNTCM	$O(\log^3 n)$	$O(n^4 \log n)$	$O(n^4)$

Table 3. Asymptotic circuit resource usage for quantum factoring of an n-bit number.

9 Conclusions and Future Work

In this paper, we have presented a 2D architecture for factoring on a quantum computer using a model of nearest-neighbor, concurrent two-qubit interactions, a classical controller, and communication between independent modules. We call this new model 2D CCNTCM. Using a combination of algorithmic improvements (carry-save adders and parallelized phase estimation) and architectural improvements (irregular two-dimensional layouts and constant-depth communication, and parallel modules), we conclude that we can run the central part of Shor's factoring algorithm (quantum period-finding) with asymptotically smaller depth than previous implementations.

A natural extension of the current work is to improve its depth to constant using the approach outlined in [36, 40], generalizing the carry-save adder to a block-save adder using threshold gates. It would also be beneficial to determine lower bounds for the time-space tradeoffs involved in Shor's factoring algorithm. These results would tell us whether we have found an optimal nearest-neighbor circuit.

Acknowledgements

The authors wish to thank Aram Harrow, Austin Fowler, and David Rosenbaum for useful discussions. P. Pham conducted the factoring part of this work during an internship at Microsoft Research. He also acknowledges funding of the architecture and layout portions of this work from the Intelligence Advanced Research Projects Activity (IARPA) via Department of Interior National Business Center contract number D11PC20167. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon. Disclaimer: The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of IARPA, DoI/NBC, or the U.S. Government.

References

1. P. Shor, "Algorithms for quantum computation: Discrete logarithms and factoring," in *Proceedings of the 35th Annual Symposium on Foundations of Computer Science*, (Santa Fe, NM), November 1994.
2. S. Beauregard, "Circuit for Shor's algorithm using 2n+3 qubits," *arXiv:quant-ph/0205095*, May 2002.
3. S. A. Kutin, "Shor's algorithm on a nearest-neighbor machine," *arXiv:quant-ph/0609001*, Aug. 2006.
4. R. Van Meter, *Architecture of a Quantum Multicomputer Optimized for Shor's Factoring Algorithm.* Ph.d., Keio University, 2006.
5. R. Van Meter and K. Itoh, "Fast quantum modular exponentiation," *Physical Review A*, vol. 71, May 2005.
6. R. Van Meter, K. Itoh, and T. Ladd, "Architecture-dependent execution time of Shor's algorithm," *Proceedings of Mesoscopic Superconductivity and Spintronics*, May 2006.
7. A. Harrow and A. Fowler, "Private communication," Oct 2011.
8. D. Rosenbaum, "Optimal Quantum Circuits for Nearest-Neighbor Architectures," *8th Conference on Theory of Quantum Computation, Communication and Cryptography*, May 2013.
9. A. Y. Kitaev, A. Shen, and M. Vyalyi, *Classical and Quantum Computation.* Providence, Rhode Island: American Mathematical Society, 2002.
10. C. Monroe, R. Raussendorf, A. Ruthven, K. R. Brown, P. Maunz, L. M. Duan, and J. Kim, "Large Scale Modular Quantum Computer Architecture with Atomic Memory and Photonic Interconnects," *arXiv:1208.0391*, Aug. 2012.
11. A. G. Fowler, "Constructing arbitrary Steane code single logical," *Quantum Information and Computation*, vol. 11, pp. 867–873, 2011.
12. A. G. Fowler, S. J. Devitt, and L. C. L. Hollenberg, "Implementation of Shor's Algorithm on a Linear Nearest Neighbour Qubit Array," *Quantum Information and Computation*, vol. 4, pp. 237–251, Feb. 2004.
13. M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information.* Cambridge, U.K.: Cambridge University Press, 2000.
14. A. Broadbent and E. Kashefi, "Parallelizing Quantum Circuits," *Theoretical Computer Science*, vol. 410, Apr. 2009.
15. D. E. Browne, E. Kashefi, and S. Perdrix, "Computational depth complexity of measurement-

based quantum computation," in *Proceedings of the Fifth Conference on Theory of Quantum Computation, Communication, and Cryptography (TQC)*, Sept. 2010.

16. R. Raussendorf, D. Browne, and H. Briegel, "Measurement-based quantum computation on cluster states," *Physical Review A*, vol. 68, Aug. 2003.
17. A. Sørensen and K. Mølmer, "Entanglement and quantum computation with ions in thermal motion," *Physical Review A*, vol. 62, p. 022311, July 2000.
18. J. Benhelm, G. Kirchmair, C. F. Roos, and R. Blatt, "Towards fault-tolerant quantum computing with trapped ions," *Nature Physics*, vol. 4, pp. 463–466, Apr. 2008.
19. P. Gossett, "Quantum Carry-Save Arithmetic," *arXiv:quant-ph/9808061*, 1998.
20. C. S. Wallace, "A Suggestion for a Fast Multiplier," *IEEE Transactions on Electronic Computers*, vol. EC-13, pp. 14–17, Feb. 1964.
21. Y. Takahashi and N. Kunihiro, "A linear-size quantum circuit for addition with no ancillary qubits," *Quantum Information and Computation*, vol. 5, no. 6, pp. 440–448, 2005.
22. Y. Takahashi, S. Tani, and N. Kunihiro, "Quantum Addition Circuits and Unbounded Fan-Out," *Quantum Information and Computation*, vol. 10, pp. 872–890, Oct. 2010.
23. D. Kielpinski, C. Monroe, and D. Wineland, "Architecture for a large-scale ion-trap quantum computer," *Nature*, vol. 417, pp. 709–711, 2002.
24. M. G. Whitney, N. Isailovic, Y. Patel, and J. Kubiatowicz, "A fault tolerant, area efficient architecture for Shor's factoring algorithm," *Proceedngs of the International Symposium on Computer Architecture*, 2009.
25. B.-S. Choi and R. Van Meter, ""$\Theta(\sqrt{n})$"-depth Quantum Adder on a 2D NTC Quantum Computer Architecture," *arXiv:1008.5093*, Aug. 2010.
26. T. G. Draper, "Addition on a Quantum Computer," *arXiv:quant-ph/0008033*, Aug. 2000.
27. V. Vedral, A. Barenco, and A. Ekert, "Quantum networks for elementary arithmetic operations," *Physical Review A*, vol. 54, pp. 147–153, July 1996.
28. C. Zalka, "Fast versions of Shor's quantum factoring algorithm," *arXiv:quant-ph/9806084v1*, 1998.
29. Y. Takahashi and N. Kunihiro, "A quantum circuit for Shor's factoring algorithm using 2n+2 qubits," *Quantum Information and Computation*, vol. 6, no. 2, pp. 184–192, 2006.
30. C. Miquel, J. Paz, and R. Perazzo, "Factoring in a dissipative quantum computer," *Physical Review A*, vol. 54, pp. 2605–2613, Oct. 1996.
31. I. García-Mata, K. Frahm, and D. Shepelyansky, "Effects of imperfections for Shors factorization algorithm," *Physical Review A*, vol. 75, p. 052311, May 2007.
32. D. Beckman, A. Chari, S. Devabhaktuni, and J. Preskill, "Efficient networks for quantum factoring," *Physical Review A*, vol. 54, pp. 1034–1063, Aug. 1996.
33. A. W. Harrow, B. Recht, and I. L. Chuang, "Efficient discrete approximations of quantum gates," *J. Math. Phys.*, vol. 43, no. 4445, 2002.
34. R. Cleve and J. Watrous, "Fast parallel circuits for the quantum Fourier transform," *arXiv:quant-ph/0006004*, June 2000.
35. C. Jones, "Distillation protocols for Fourier states in quantum computing," *arXiv:1303.3066*, Mar. 2013.
36. P. Høyer and R. Špalek, "Quantum Circuits with Unbounded Fan-out," *Theory of Computing*, vol. 1, pp. 81–103, Aug. 2005.
37. M. Amy, D. Maslov, M. Mosca, and M. Roetteler, "A meet-in-the-middle algorithm for fast synthesis of depth-optimal quantum circuits," *arXiv:1206.0758*, June 2012.
38. C. Jones, "Low-overhead constructions for the fault-tolerant Toffoli gate," *Physical Review A*, vol. 87, p. 022328, Feb. 2013.
39. T. G. Draper, S. A. Kutin, E. M. Rains, and K. M. Svore, "A logarithmic-depth quantum carry-lookahead adder," *Quantum Information and Computation*, vol. 6, pp. 192–212, June 2006.
40. K.-Y. Siu, J. Bruck, T. Kailath, and T. Hofmeister, "Depth efficient neural networks for division and related problems," *IEEE Transactions on Information Theory*, vol. 39, pp. 946–956, May 1993.