Quantum Information and Computation, Vol. 0, No. 0 (2003) 000–000

A 2D Nearest-Neighbor Quantum Architecture for Factoring

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 (received date)
Revised (revised date)

We present a 2D nearest-neighbor quantum architecture for Shor's factoring algorithm in polylogarithmic depth. Our implementation uses parallel phase estimation, constant-depth fanout and teleportation, and constant-depth carry-save modular addition. We derive asymptotic bounds on the circuit depth and width of our architecture and 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 width.

Keywords: quantum architecture, prime factorization, Shor's algorithm, nearest-neighbor, carry-save addition
Communicated by: to be filled by the Editorial

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 of prime integer factorization on a two-dimensional architecture that allows concurrent (parallel) two-qubit operations between neighboring qubits. We show that our circuit construction is asymptotically more efficient in circuit depth than previous state-of-the-art techniques, achieving a depth of $O(\log^2 n)$ and a width of $O(n^4)$ qubits, as detailed in Table 20 of Section 8.

Our technique hinges on several key building blocks, which we describe in sequence. 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

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

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 show how to use as a basis for a modular multiplier (Section 6) and a modular exponentiator (Section 7). Finally, we analyze the asymptotic circuit resources required by our approach and compare them to previous implementations.

2 Background

Quantum architecture is concerned with the physical layout of qubits and constraints on their interactions, as well as the efficient execution, in time, space, and other resources, of algorithms on a given architecture. In this paper, we focus on designing a realistic nearest-neighbor circuit for running Shor's factoring algorithm on architectural models of a physical quantum device.

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 *implementation*, 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.

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.

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 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, so that theoretically 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. 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, with definitions equivalent to those in [8].

Definition 1 *A 2D CCNTC architecture consists of*

- *a planar graph* (V, E) *where* V *represents all qubits and* E *represents allowed two-qubit interactions between qubits*
- *a universal gate set* $X, Z, H, T, T^\dagger, CNOT$

- *a deterministic machine (classical controller) M that applies*

At

The third extension, 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.

2.2 2D CCNTCM*: Two-Dimensional Nearest-Neighbor Two-Qubit Concurrent Gates with 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 (citation needed, scalable NIST architecture, or MUSIQC architecture). We call these individual machines *modules*, each of which is a self-contained 2D CCNTC lattice. We treat these modules and allowed 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 using of the omnipresent classical controller. The modules can be arbitrarily far apart and have arbitrary connectivity with other modules. Teleportations can occur concurrently between disjoint modules, just as operations can occur concurrently on disjoint qubits in NTC. 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 1. Each module contains within it a 2D CCNTC lattice.

We measure the efficiency of a circuit on a particular architecture in terms of three conventional resources and three new resources based on modules:

circuit size the number of non-identity gates

circuit depth the number of time-steps allowing parallel operations

circuit width the total number of qubits, including inputs, outputs, and ancillae

module size the number of qubits teleported between modules

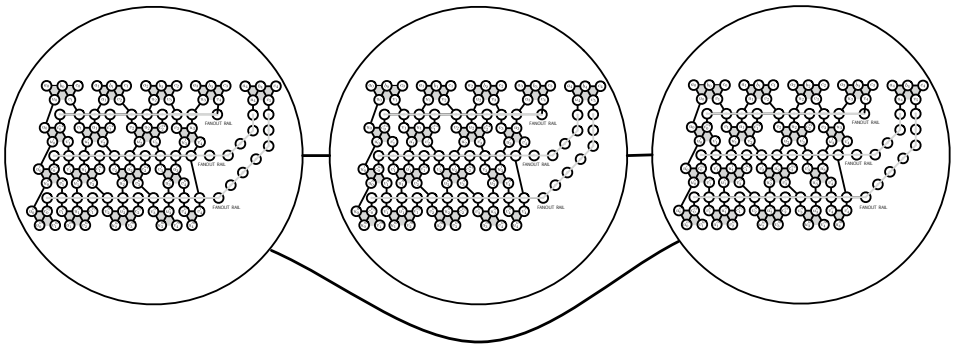

Fig. 1. Three modules in the 2D CCNTCM model

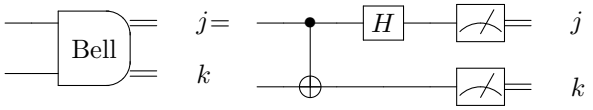

Fig. 2. A circuit for measurement in the Bell state basis.

Depth	Size	Width
7	3n + 4	n+1

Fig. 3. Circuit resources for the constant-depth teleportation circuit shown in Figure 4 for transporting a single qubit across n qubits.

module depth the depth of consecutive teleportations between modules

module width the number of modules

We count the following single-qubit and two-qubit operations as having unit size and unit depth: the generators of the two-qubit Clifford group (the Pauli X gate, the Pauli Z gate, the Hadamard (H) gate, and the controlled NOT (CNOT) gate), the T gate and its inverse $T^\dagger$, and single-qubit measurements (in the Z-basis). Together, all of these except the last form a universal set of unitary gates [9].

A measurement in the Bell basis we interpret as having depth 4, size 4, and width 2, of which two of those are reusable according to the circuit in Figure 2

This 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 in a fault-tolerant way, especially the T gate which cannot be performed transversally in many popular error-correcting codes [10].

In both models, multiple gates acting on disjoint qubits can occur in parallel during the same timestep.

2.3 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 at one site to another over arbitrarily long distances can be addressed by using the constant-depth teleportation circuit shown in Figure 4, illustrated using standard quantum circuit notation [11]. This uses the circuit resources shown in Table 3. The depth includes a layer of H gates; two interleaved layers of CNOTs; a layer of Bell basis measurement; 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 $|0\rangle$.

Although general cloning is impossible [11], the second problem of copying information can be addressed by performing an unbounded quantum fanout operation: $|x, y_1, \ldots, y_n\rangle \rightarrow$

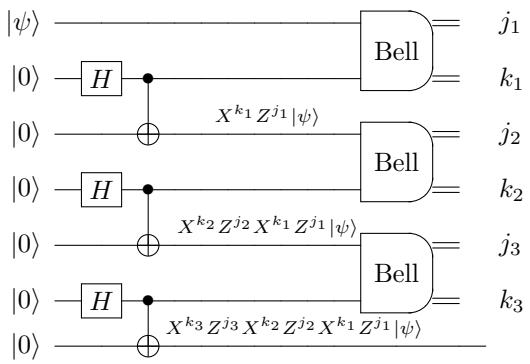

Fig. 4. Constant-depth circuit based on [12, 13] for teleportation over $n = 5$ qubits [8].

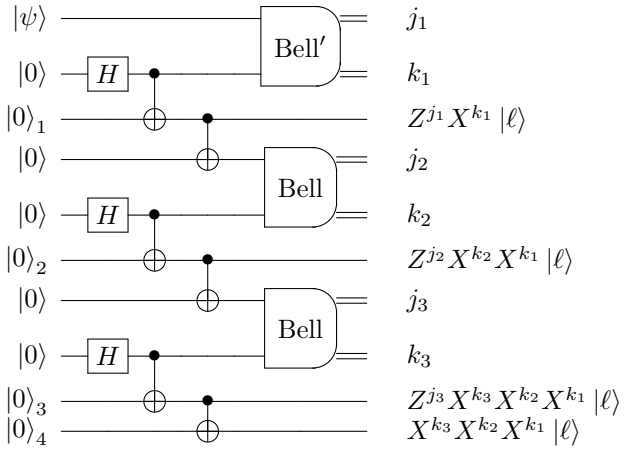

Fig. 5. Constant-depth circuits based on [12, 13] for fanout [7] of one qubit to $n = 4$ entangled copies.

$|x, y_1 \oplus x, \ldots, y_n \oplus x\rangle$. 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 [14] that relies on the creation of an n-qubit cat state [13].

It 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 Figure 6. The constant-depth fanout circuit is shown in Figure 5 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

Depth	Size	Width
9	10n - 9	3n-1

Fig. 6. Circuit resources for the constant-depth fanout circuit shown in Figure 5 for creating n entangled copies of a single qubit.

Depth	Size	Width
$8\log_2(2n)$	$33n\log_2(2n) + 10\log_2^2(2n)$	$3n-1$

Fig. 7. Circuit resources for un-fanout circuit consisting of alternating rounds of constant-depth teleportation and CNOT.

into the larger fanned out state given in Equation 1. 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 beginning in state $|0\rangle_1$, $|0\rangle_2$, $|0\rangle_3$, and $|0\rangle_4$). The Pauli corrections X^{k_1} and Z^{j_1} are from the Bell basis measurement entangling the cat state with the source qubit (denoted Bell$'$).

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

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 open problem. In this work it is sufficient to perform 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 7.

3 Related Work

Our work builds upon ideas in classical digital and reversible logic and their extension to quantum logic. A 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) [15]. The techniques relies on encoded addition, sometimes called a 3-2 adder, and derives from classical Wallace trees [16].

Takahashi and Kunihiro discovered a linear-depth and linear-size quantum adder using zero ancillae [17]. They also developed an adder with tradeoffs between $O(n/d(n))$ ancillae and $O(d(n))$-depth for $d(n) = \Omega(\log n)$ [18], which has better width at the cost of worse depth than our present work. 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 by early ion trap proposals [19], and later analyzed at the level of physical qubits and error correction in the context of Shor's algorithm [20]. 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 [21] 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 relies on the underlying adder implementation. Since Shor's algorithm is a probabilistic algorithm, requiring several rounds of QPF to amplify success probability, it suffices to determine the resources required for a single round of QPF with a fixed, modest success probability.

The most common approach to QPF performs controlled modular exponentiation followed by an inverse quantum Fourier transform (QFT) [11]. We will call this *serial QPF*. Beauregard [2] uses the serial QPF approach to construct a cubic-depth quantum period-finder using only $2n+3$ qubits on AC, by combining the ideas of Draper's transform adder [22], Vedral et al.'s modular arithmetic blocks [23], and a semi-classical QFT. This approach was subsequently adapted to 1D NTC by Fowler, Devitt, and Hollenberg [24] to achieve exact 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, also in quadratic depth [25]. Takahashi and Kunihiro extended their earlier $O(n)$-depth adder to a factoring circuit in $O(n^3)$-depth with linear width [26].

In these 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 and allows access to all qubits in parallel. Therefore, we optimize circuit depth at the expense of width. We primarily compare our work primarily to Kutin's method [3].

These works also rely on serial QPF, which requires 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, which requires at least a $O(\log(1/\epsilon))$ overhead in depth to approximate a rotation with precision ϵ [27, 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 [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 20 of Section 8.

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 [13]. Their model, which we adapt and call CCNTC, uses the classical controller mentioned in 2.3. They describe a constant-depth circuit for exact factoring,

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

Fig. 8. An example of carry-save encoding for the 5-bit conventional number 30.

improving on a constant-depth circuit for approximate factoring by Høyer and Špalek [28]. A direction for future work is to determine how our approach compares to the BKP result in terms of circuit size and width.

4 The Constant-Depth Carry-Save Technique

Our 2D factoring approach rests on the central technique of the constant-depth carry-save adder (CSA) [15], 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 conventional numbers, u and v. We say $(u+v)$ is a *carry-save encoded*, or CSE, number. The CSE representation itself 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 8 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:

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

Using this notation, the following identity holds:

$$x_{(j,k)} = x_{(j,\ell)} + x_{(\ell+1,k)}, \qquad \text{for all } j \le \ell < k. \tag{3}$$

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

$$x = x_{(0,n-1)} \equiv u + v = u_{(0,n-2)} + v_{(1,n-1)}. \tag{4}$$

Finally, we denote arithmetic modulo m, $x_{(j,k)} \bmod m$, as $x_{(j,k)}[m]$.

Figure 9 gives a circuit description of carry-save addition (CSA) for a single bit position i. The resources for this circuit are given in Figure 10, using the resources for the Toffoli gate in Figure 12.

We must layout the circuit to satisfy a 2D NTC model. The Toffoli gate decomposition in [29], duplicated in Figure 11, requires two control qubits and a single target qubit to be

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

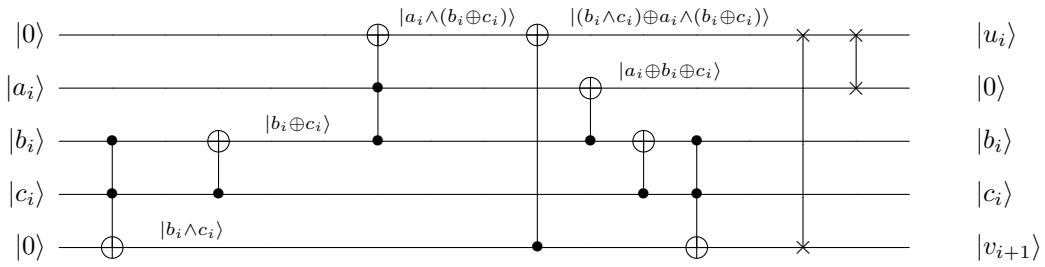

Fig. 9. Carry-save adder circuit for a single bit position i: $a_i + b_i + c_i = u_i + v_{i+1}$.

Depth	Size	Width
33	55	5

Fig. 10. Circuit resources for the carry-save adder circuit from Figure 9.

mutually connected to each other. Given this constraint, and the interaction of the CNOTs in Figure 9, we can rearrange these qubits on a 2D planar grid and obtain the layout shown in Figure 13, which satisfies our 2D NTC model. Qubits $|a\rangle_i$, $|b\rangle_i$, and $|c\rangle_i$ reside at the top of Fig. 13, 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 Fig. 9. Note that this construction uses more gates and one more ancilla than the equivalent quantum full adder circuit in Figure 5 of [15], however this is necessary in order to meet our architectural constraints and does not change the asymptotic results. Also in Figure 13 is a variation called a 2-2 adder, which simply re-encodes two i-bits into an i-bit and an $(i+1)$-bit, which 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. 9). 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.

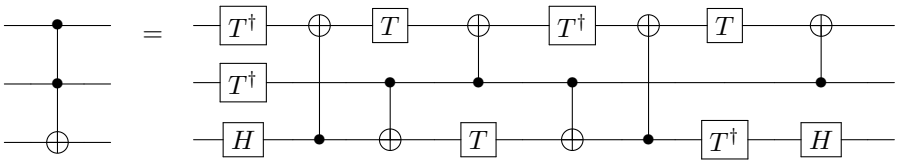

Fig. 11. The depth-efficient Toffoli gate decomposition from [29].

Depth	Size	Width
8	15	3

Fig. 12. Circuit resources for the Toffoli circuit given in Figure 11.

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 [15]. 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 $x_{(n-1,n)}[m]$:

$$\begin{aligned} x \bmod m &= x_{(0,n)}[m] \\ &= x_{(0,n-2)} + x_{(n-1,n)}[m]. \end{aligned} \tag{5}$$

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 $2n-3$ bits: $\{u_0, u_1, \ldots, u_{n-2}\} \cup \{v_1, v_2, \ldots, v_{n-2}\}$. Then we add back the three modular residues $(v_{(n)}[m], u_{(n-1)}[m], v_{(n-1)}[m])$, and we are guaranteed not to generate additional overflow bits (of significance 2^{n-1} or higher). This equivalence is

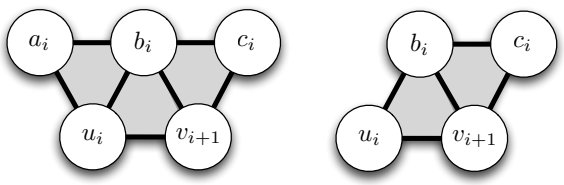

Fig. 13. The carry-save adder (CSA), or 3-2 adder, and carry-save 2-2 adder.

shown in Eq 6.

$$\begin{aligned}(u+v)[m] &= \left(u_{(0,n+1)} + v_{(1,n+2)}\right)[m] \\ &= u_{(0,n)} + v_{(1,n)} + \\ &\quad u_{(n+1)}[m] + v_{(n+1)}[m] + \\ &\quad v_{(n+2)}[m] \end{aligned} \tag{6}$$

Lemma 1 (Modular Reduction in Constant Depth [15]) *The modular addition of three n-bit numbers to two n-bit numbers can be accomplished in constant depth.*

Proof: Our goal is to show how to perform modular addition while keeping our numbers of a fixed size by treating overflow bits correctly. 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 14.

	a_4	a_3	a_2	a_1	a_0	5-bit input number a
	b_4	b_3	b_2	b_1	b_0	5-bit input number b
	c_4	c_3	c_2	c_1	c_0	5-bit input number c
	u_4	u_3	u_2	u_1	u_0	truncate u_4
v_5	v_4	v_3	v_2	v_1		truncate v_4, v_5
			$c_2^{v_4}$	$c_1^{v_4}$	$c_0^{v_4}$	add back 2^4 mod m controlled on v_4
		u'_3	u'_2	u'_1	u'_0	
	v'_4	v'_3	v'_2	v'_1		
			$c_2^{u_4}$	$c_1^{u_4}$	$c_0^{u_4}$	add back 2^4 mod m controlled on u_4
	u''_4	u''_3	u''_2	u''_1	u''_0	the bit u''_4 is the same as v'_4
	v''_4	v''_3	v''_2	v''_1		
			$c_2^{v_5}$	$c_1^{v_5}$	$c_0^{v_5}$	add back 2^5 mod m controlled on v_5
	u'''_4	u'''_3	u'''_2	u'''_1	u'''_0	Final CSE output with 5 bits
	v'''_4	v'''_3	v'''_2	v'''_1		Final CSE output with 5 bits

Fig. 14. A schematic proof of Gossett's constant-depth modular reduction for $n = 3$.

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 a sufficient condition for no v'''_{n+2} bit being generated in the third modular reduction layer. This bit is the majority of u''_{n+1}, v''_{n+1}, and $c_{n+1}^{v_{n+2}} = 0$. This means we only have to guarantee that at most one of u''_{n+1} and v''_{n+1} has value 1. This is equivalent to requiring that $u''_{(n,n+1)} + v''_{(n+1)} \leq 3 \cdot 2^n$, that is, the sum of these three bits has value at most 3. Bit u''_{n+1} is copied directly from v'_{n+1} by the rules of CSA, which requires the following condition for the second modular reduction layer: $u'_{(n)} + v'_{(n,n+1)} \leq 3 \cdot 2^n$. This is true because $u'_{(n)} + v'_{(n+1)} = u_{(n)} + v_{(n)} \leq 2$ and $v'_{(n)} \leq 1$. Everywhere we use the fact that the modular residues are restricted to n bits. Therefore, 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 simplicity.

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.

5.1 A Concrete Example of Modular Addition

A 2D circuit for modular addition of 5-bit numbers using four layers of parallel CSA's is shown graphically in Figure 15 which corresponds directly to the schematic proof in Figure 14. Figure 15 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.

In layer 2, we use a constant-depth fanout rail (see Figure 5) 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 4.

Layer 3 operates similarly to layer 2, except that the modular residue is controlled on u_4:

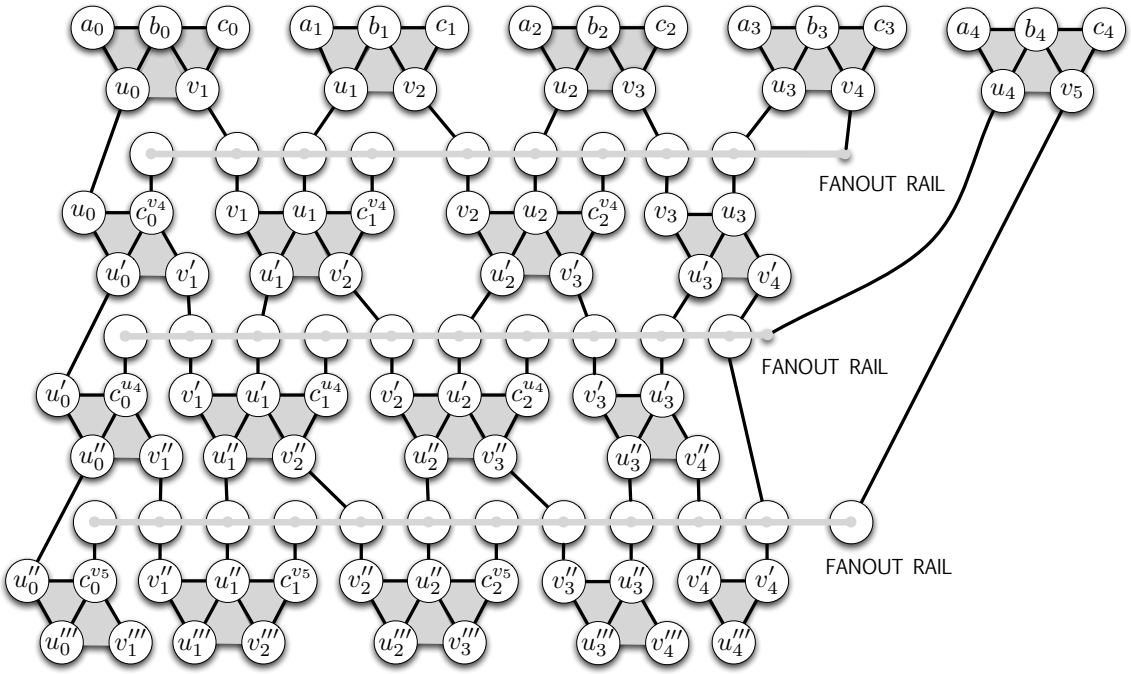

Fig. 15. Addition and three rounds of modular reduction for a 3-bit modulus.

$|c^{u_4}\rangle \equiv |2^4[m] \cdot u_4\rangle$. 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: $|c^{v_5}\rangle \equiv |2^5[m] \cdot v_5\rangle$. 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, illustrated in Figure 15 for $n = 3$, 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 16 for the rest of this paper.

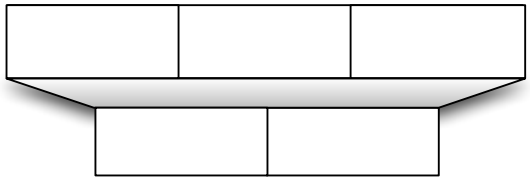

Fig. 16. Symbol for an n-bit 3-to-2 modular adder, also called a CSA tile.

Depth	Size	Width
356	572n + 724	33n+47

Fig. 17. Circuit resources for the n-bit modular 3-to-2 adder represented by Figure 15.

5.2 Quantum Circuit Resources for Modular Addition

We now calculate numerical upper bounds for the circuit resources of the $(n+2)$-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 Figure 10. 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 15, 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 detailed resources are listed in Figure 17, which is twice the resources described in the previous two paragraphs due to reversing the computation. The $3n$ garbage qubits left over are the fanned out copies of the overflow bits. It is not currently known how to uncompute these in constant depth. However, we can uncompute them in logarithmic depth using alternating layers of constant-depth teleportation and CNOT, and we can this for all CSA tile layers in parallel during modular multiplication (see the next section), where the logarithmic depth is subsumed by the modular multiplier itself.

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 6.1. Both of these problems use as a subroutine the generic problem of *modular multiple addition* which we define and solve in 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: $z^{(i)} = 2^i a b_i \bmod m$. 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.

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 number modulo m, which is described in 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.

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, but using a more detailed procedure given in Section 6.2. 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.

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. We do not argue that our construction is optimal, but this is an interesting open problem for future work.

Our construction is as follows. 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

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

are at least $2n^2 - 2n + 1$ such single-bit partial products, which can be grouped into at most $2n + 3$ 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.

$$t = 2n^2 + 16n + 11 \tag{7}$$

The creation of the product bits $|x_i \cdot y_j\rangle$ occurs on a square lattice, with the bits x_i and y_j on adjacent edges. This takes place in 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 $\lceil \log_2(2n + 3) \rceil$ 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 site, out of the square lattice, to a different location where the $\left|z^{(i,j)}\right\rangle$ numbers are later generated. We will call these locations 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 any product bits $x_i \cdot y_j$ as in the last round, which is half the total number of $(2n + 3)^2$.

The z-sites have total width of $3nt'$ qubits, so the maximum total teleportation length of all qubits is $(2n + 3)^2$ multiplied by the maximum length of any single teleportation length, Our construction consists of the following steps.

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 $((2n + 3)(n + 2))^2$.

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.

5. Perform modular multiple addition (described in Section 6.3) on t' n-qubit quantum integers down to 2 n-qubit quantum integers.

6. Uncompute all the previous steps to restore ancillae to $|0\rangle$.

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:

$$z^{(1)} + z^{(2)} + \ldots z^{(t')} \equiv (u + v)[m]. \tag{8}$$

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 $\lceil \log_{3/2}(t'/3) \rceil + 1$.

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 18. The module for each CSA tile is represented by the symbol from Figure 16. The arrows indicate the teleportation of output numbers from the source tile to be input numbers into a destination tile.

Now we can analyze the circuit resources for multiplying n-bit quantum integers, which requires $(t' - 2)$ modular additions, for $t' = n^2$. The circuit width is the sum of the $O(n^3)$ ancillae needed for number generation 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:

$$D_{MM} = 2205 \log_2 n + 6066 \tag{9}$$

The module depth is:

$$\overline{D}_M M = 2 \log_2 n + 11 \tag{10}$$

The circuit size is:

$$\begin{aligned} S_{MM} = \quad & (12n^2 + 644n + 288) \log_2^2(10n) + & (11) \\ & (74n^3 + 606n^2 + 292n + 41) \log_2^2(10n) + & (12) \\ & (1228n^3 + 10151n^2 + 14397n + 4645) & (13) \end{aligned}$$

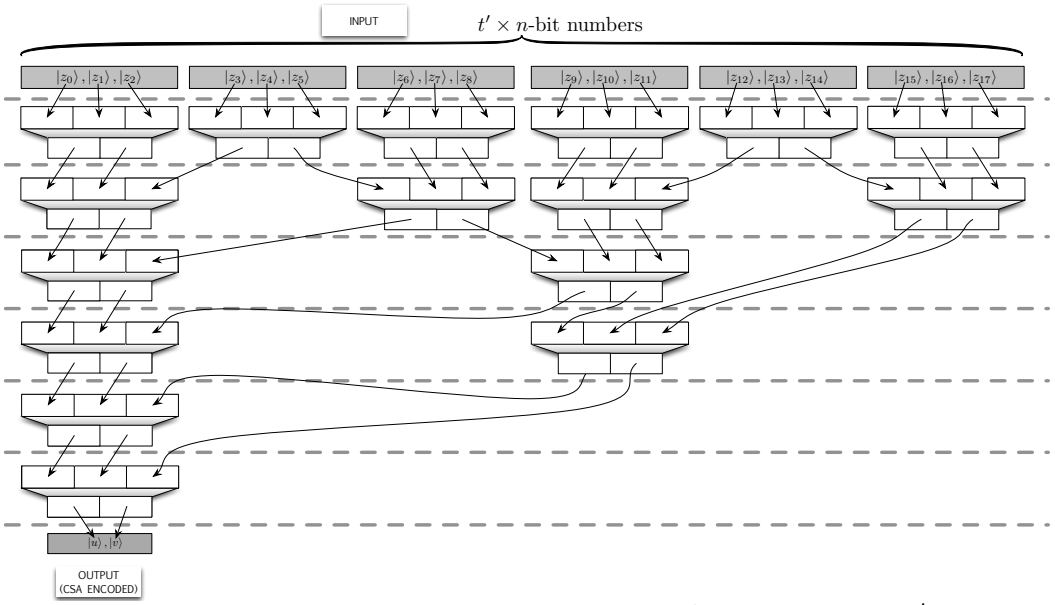

Fig. 18. Modular multiple addition of quantum integers on a CSA tile architecture for $t' = 18$ in a logarithmic-depth tree with height $(\lceil \log_{\frac{3}{2}}(t'/3) \rceil + 1) = 6$.

The module size is:

$$\overline{S}_{MM} = 8n^4 + 86n^3 + 276n^2 + 325n + 115 \tag{14}$$

The circuit width is:

$$W_{MM} = 66n^3 + 558n^2 + 870n + 290 \tag{15}$$

The module width is

$$\overline{W}_{MM} = 4n^2 + 28n + 15 \tag{16}$$

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 a n-qubit quantum input number $|x\rangle$ by t classical numbers $a^{(j)}$, we can multiply them in series. This requires depth $O(t \log n)$ based on the modular multipliers in previous sections.

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 19, where arrows indicate multiplication. The tree has depth $\log_2(t)$ in modular multiplier operations. Furthermore, each modular multiplier operation has depth $O(\log(n))$ for n-qubit numbers. Therefore, the

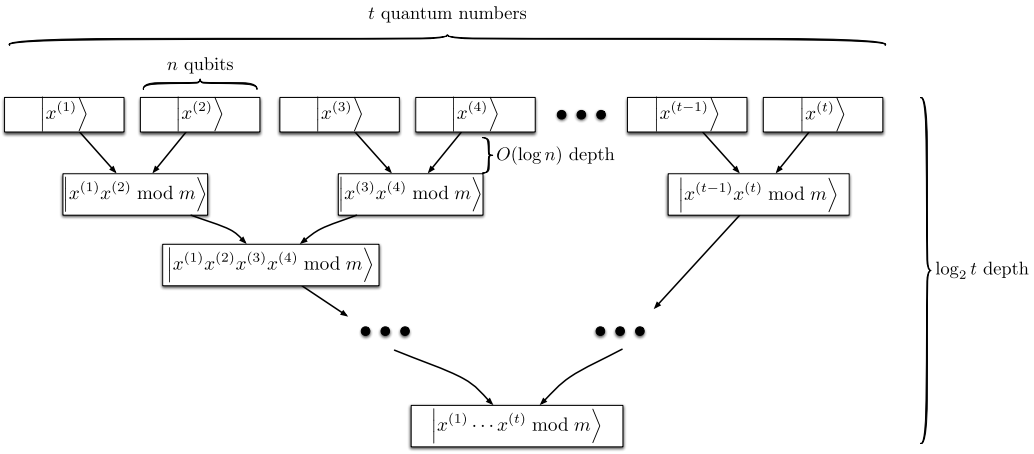

Fig. 19. Parallel modular exponentiation: multiplying t quantum integers in a $O(\log(t)\log(n))$-depth binary tree.

overall depth of this parallel modular exponentiation structure is $O(\log(t)\log(n))$. In phase estimation for QPF, it is sufficient to take $t = O(n)$ [11, 9]. Therefore our total depth is $O(\log^2(n))$ as desired. At this point, combined with the parallel phase estimation procedure of [9], we have a complete factoring implementation in our 2D nearest-neighbor architecture.

For all known QPF procedures, there are $t = O(n)$ control bits needed, and also $O(n)$ modular multiplications in a tree of depth $O(\log n)$. Each modular multiplication has depth $O(\log n)$ and width $O(n^3)$. Therefore, the depth of the parallel modular exponentiation circuit above is $O(\log^2 n)$ and the width is $O(n^4)$. 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.

This leads to the following circuit resources for a modular exponentiator.

The circuit depth is:

$$D_{ME} = 2205 \log_2^2 n + 33586 \log_2 n + 75722 \tag{17}$$

The module depth is:

$$\overline{D}_{ME} = \log_2 n + 13 \tag{18}$$

The circuit size is:

$$S_{ME} = 22933n^5 + 244609n^4 + 789182n^3 + 930414n^2 + 329335n - 155 \tag{19}$$

The module size is:

$$\overline{S}_{ME} = 5734n^2 + 8598n - 3 \tag{20}$$

The circuit width is:

$$W_{ME} = 94598n^4 + 799749n^3 + 1246692n^2 + 415222n - 145 \tag{21}$$

The module width is:

$$\overline{W}_{ME} = 1434n \tag{22}$$

8 Asymptotic Results

The resources required for our approach, as well as other nearest-neighbor approaches, are listed in Table 20, where the asymptotic resource bounds assume some fixed constant error probability for each round of period-finding. 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 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, KSV-style modular exponentiator. Our approach is the first 2D NTC implementation for factoring.

Implementation	Architecture	Depth	Size	Width
Vedral, Barenco & Ekert [23]	AC	$O(n^3)$		$O(n)$
Gossett [15]	AC	$O(n \log n)$		$O(n^2)$
Beauregard [2]	AC	$O(n^3)$		$O(n)$
Zalka [25]	AC	$O(n^2)$		$O(n)$
Takahashi & Kunihiro [26]	AC	$O(n^3)$		$O(n)$
Fowler, Devitt, Hollenberg [24]	1D NTC	$O(n^3)$		$O(n)$
Kutin [3]	1D NTC	$O(n^2)$		$O(n)$
Current Work	2D CCNTCM	$O(\log^2 n)$		$O(n^4)$

Fig. 20. Asymptotic 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 combination of algorithmic improvements (carry-save adders and parallelized phase estimation) and architectural improvements (irregular two-dimensional layouts and constant-depth communication), we conclude that we can run the central part of Shor's factoring algorithm (quantum period-finding) with asymptotically smaller depth than previous implementations.

For future work, we would like to determine the exact width, depth, and size of our proposed factoring circuit, including the constants, as well as further optimizing our depth to be constant. Along those lines, Rosenbaum recently showed how to convert any n-qubit CCAC circuit to an equivalent CCNTC circuit in constant depth using n^2 ancillae [8]. It is an interesting open question how a generic conversion of a constant-depth CCAC factoring architecture [13, 28] to CCNTC compares to our hand-optimized circuit. The depth of our

circuit may also be improved by extending the carry-save adder from a $3 \to 2$ circuit to any $2^{n-1} \to n$ 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." May 2002.
3. S. A. Kutin, "Shor's algorithm on a nearest-neighbor machine." 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." Apr. 2012.
9. M. V. A. Yu. Kitaev, A.H. Shen, *Classical and Quantum Computation*. Providence, Rhode Island: American Mathematical Society, 2002.
10. A. G. Fowler, "Constructing arbitrary Steane code single logical," *Quantum Information and Computation*, vol. 11, pp. 867–873, 2011.
11. M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information*. Cambridge, U.K.: Cambridge University Press, 2000.
12. A. Broadbent and E. Kashefi, "Parallelizing Quantum Circuits," *Theoretical Computer Science*, vol. 410, Apr. 2009.
13. 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.
14. R. Raussendorf, D. Browne, and H. Briegel, "Measurement-based quantum computation on cluster states," *Physical Review A*, vol. 68, Aug. 2003.
15. P. Gossett, "Quantum Carry-Save Arithmetic." 1998.
16. C. S. Wallace, "A Suggestion for a Fast Multiplier," *IEEE Transactions on Electronic Computers*, vol. EC-13, pp. 14–17, Feb. 1964.
17. 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.
18. 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.
19. D. Kielpinski, C. Monroe, and D. Wineland, "Architecture for a large-scale ion-trap quantum computer," *Nature*, vol. 417, pp. 709–711, 2002.
20. 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.
21. B.-S. Choi and R. Van Meter, ""$\Theta(\sqrt{n})$"-depth Quantum Adder on a 2D NTC Quantum Computer Architecture." Aug. 2010.
22. T. G. Draper, "Addition on a Quantum Computer." Aug. 2000.
23. V. Vedral, A. Barenco, and A. Ekert, "Quantum networks for elementary arithmetic operations," *Physical Review A*, vol. 54, pp. 147–153, July 1996.
24. 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.
25. C. Zalka, "Fast versions of Shor's quantum factoring algorithm." 1998.
26. 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.
27. I. L. C. Aram W. Harrow, Benjamin Recht, "Efficient discrete approximations of quantum gates," *J. Math. Phys.*, vol. 43, no. 4445, 2002.
28. P. Høyer and R. Špalek, "Quantum Circuits with Unbounded Fan-out," *Theory of Computing*, vol. 1, pp. 81–103, Aug. 2005.
29. M. Amy, D. Maslov, M. Mosca, and M. Roetteler, "A meet-in-the-middle algorithm for fast synthesis of depth-optimal quantum circuits," June 2012.