# A 2D Nearest-Neighbor Quantum Architecture for Factoring

Paul Pham
University of Washington
Quantum Theory Group
Box 352350, Seattle, WA 98195, USA,
ppham@cs.washington.edu,
http://www.cs.washington.edu/homes/ppham/

Krysta M. Svore
Microsoft Research
Quantum Architectures and Computation Group
One Microsoft Way, Redmond, WA 98052, USA
ksvore@microsoft.com,
http://research.microsoft.com/en-us/people/ksvore/

July 27, 2012

## Abstract

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.

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

## 1 Introduction

Shor's factoring algorithm is a central result in quantum computing, with an exponential speed-up over the best-known classical algorithm [16]. 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, 11, 20, 21, 22]. 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 novel quantum architecture for prime integer factorization in two dimensions that allows concurrent (parallel) two-qubit operations between neighboring qubits.

Our paper is organized as follows. Section 2 introduces quantum architectural models, circuit resources, and constant-depth communication techniques due to [9, 15]. Section 3 places this 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 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). Finally, we analyze the asymptotic circuit resources required by our approach and compare them to previous implementations in the related work.

# 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 [21], 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, which 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 extend to a two-dimensional regular grid (2D NTC), where each qubit has four neighbors and there is an extra degree of freedom in which to move data. In this paper, we extend the 2D NTC model in two 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 we make is the realistic assumption that classical control can access every qubit in parallel, and we do not count these classical resources in our implementation. We call these augmented models CCAC and CCNTC following [15]. The classical controller corresponds to fast digital computers which are available in actual experiments and are necessary for constant-depth communication in the next section.

We measure the efficiency of a circuit on a particular architecture in terms of three resources: *circuit size* (number of non-identity gates), *circuit depth* (number of time-steps), and *circuit width* (number of qubits). For circuit depth, a two-qubit gate takes one time-step and absorbs any adjacent single-qubit gates. Multiple two-qubit gates on disjoint qubits can occur in parallel during the same timestep.

## 2.2 Constant-depth Teleportation and Fanout

Two key problems in nearest-neighbor architectures deal with communication, namely moving and copying quantum information. How can we transport quantum information at one site to another over arbitrarily long distances? To solve this problem, we employ the constant-depth teleportation circuit shown in part (a) of Figure 1, using standard quantum circuit notation from [12].

The second problem is copying information. Although general cloning is impossible [12], we only need to perform unbounded quantum fanout, the 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

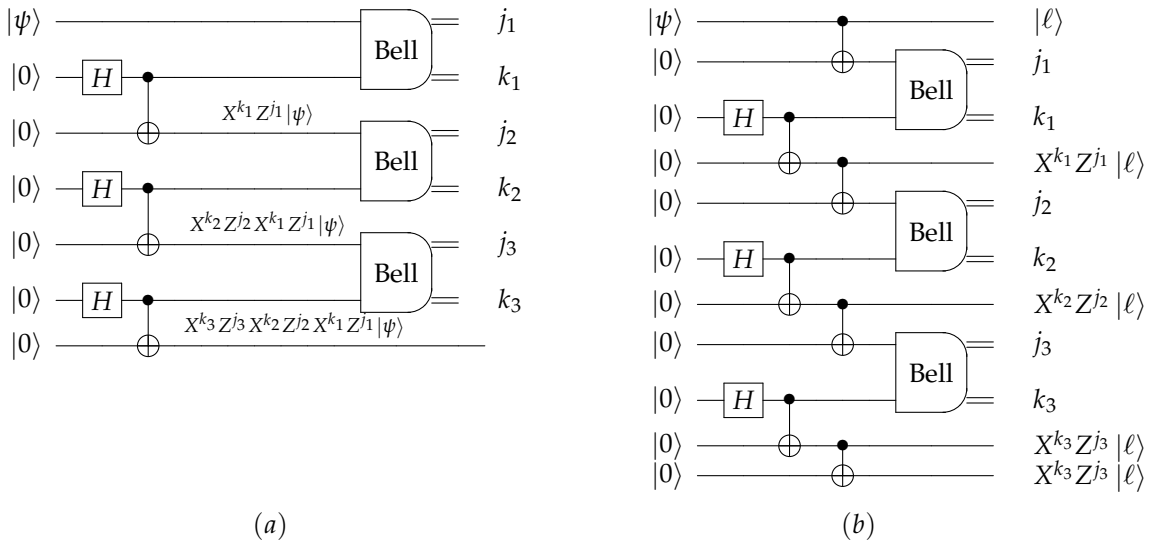


Figure 1: Constant-depth circuits based on [3, 4] for (a) teleportation [15] and (b) fanout [9].

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 [4]. It requires $O(1)$-depth, $O(n)$-size, and $O(n)$-width, and is shown in part (b) of Figure 1 for the case of fanning out $|\psi\rangle$ to four qubits. The technique works by creating multiple small cat states of a fixed size (in this case, three qubits) and linking them together with Bell measurements. The qubits marked $|\ell\rangle$ are entangled into a (slightly) larger cat state, up to Pauli corrections.

$$\frac{1}{\sqrt{2}} X_1^{k_1} Z_1^{j_1} X_2^{k_2} Z_2^{j_2} X_3^{k_3} X_4^{k_3} Z_3^{j_3} Z_4^{j_3} \left(|0000\rangle + |1111\rangle\right) \tag{1}$$

The operators $X_i^k$ and $Z_\ell^j$ denote Pauli $X$ and $Z$ operators on qubits $i$ and $\ell$, controlled by classical bits $k$ and $j$, respectively. These corrections are enacted by the classical controller based on the Bell measurement outcomes (not depicted). Unfortunately, this "consumes" the cat state in that there is no known way to unentangle the source qubit from the cat state after they have been jointly measured [15].

# 3 Related Work

We extend the body of work which applies classical ideas to quantum logic. Gossett [8] uses carry-save techniques to add numbers in constant-depth and multiply in logarithmic-depth using a special encoding, but at a quadratic cost in qubits (circuit width). The underlying idea of encoded adding, sometimes called a 3-2 adder, derives from Wallace trees [24].

Choi and Van Meter are the first to discuss 2D architectures by designing an adder that runs in $\Theta(\sqrt{n})$-depth on 2D NTC [5] using $O(n)$-qubits with dedicated, special-purpose areas of a physical circuit layout.

Takahashi and Kunihiro have also discovered a linear-depth and linear-size adder using zero ancillae [17], and also an adder with variable tradeoffs between $O(n/d(n))$ ancillae and $O(d(n))$-depth for $d(n) = \Omega(\log n)$ [19] which has better width

but worse depth than our adder. This approach assumes unbounded fanout, which has not been mapped to a nearest-neighbor circuit until the current work.

Once an adder implementation is chosen, it can be extended to perform modular reduction, modular multiplication, modular exponentiation, and ultimately quantum period finding (QPF), the only quantum part of the factoring algorithm. 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 original approach to QPF performs controlled modular exponentiation followed by an inverse quantum Fourier transform (QFT) [12]. We will call this *serial QPF*.

This is the approach taken by all other factoring (QPF) implementations on any architectural model before the current work. For example, Beauregard [2] uses this 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 [6], 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 [7] to achieve exact resource counts for an $O(n^3)$-depth quantum period-finder. Kutin [11] later improved this using an idea from Zalka for approximate multipliers to get 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 Tani extend their earlier $O(n)$-depth adder to a factoring circuit in $O(n^3)$-depth but with linear width.

All these works assume qubits are expensive (width) and that execution time (depth) is not the limiting constraint. We compare our work primarily against Kutin's method, and we make the alternative assumption that ancillae are cheap and that fast classical control is available which can access all qubits in parallel. Therefore, we optimize circuit depth at the expense of width.

Serial QPF is depth-limited by having to the perform 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 $O(n \log n)$ for factoring $n$-bit numbers. There is an alternative, parallel version of phase estimation described in Section 13 of [1], which decreases depth in exchange for increased width and additional classical post-processing. This eliminates the need to do an inverse QFT. We refer the reader to [1] and [13] for details. Our factoring scheme employs our 2D quantum arithmetic circuits and this *parallel QPF*, and we will show that it is asymptotically more efficient than the other QPF method. We compare the circuit resources required by our work with the serial QPF implementations above in Table 1 of Section 8.

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 [4]. Their model, which we adapt and call CCNTC, uses the classical controller mentioned in 2.2. They describe a constant-depth circuit for exact factoring, improving on a constant-depth circuit for approximate factoring by Høyer and Špalek [10]. 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) [8], which converts the sum of three numbers $a$, $b$, and $c$, to the sum of two numbers $u$ and $v$: $a + b + c = u + v$. To explain this technique and how it achieves constant depth, we need 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.[1] Equivalently, $x$ can be represented as a (non-unique) sum

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

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.

At the level of bits, 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 $\{a_i, b_i, c_i\}$. See Figure 2 for 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 will denote taking the modular residue of a number as follows: $x_{(j,k)}[m] \equiv x_{(j,k)} \bmod m$.

Using a Toffoli gate decomposition (see p. 182 [12]), two control qubits and a single target qubit must be mutually connected to each other. Given this constraint, and the interaction of the CNOTs in Figure 3, we can rearrange these qubits on a 2D planar grid and obtain the layout shown in Figure 4, which satisfies our 2D NTC model. Note that this uses more gates and one more ancilla than the equivalent quantum full adder circuit in Figure 5 of [8], but this is necessary to meet our architectural constraints and does not change the asymptotic results. Also in Figure 4 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's. Since each CSA operates in constant depth, the entire layer also operates in constant-depth, and we have achieved (non-modular) addition.

An important consideration here is the circuit width. The circuit above operates out-of-place and produces two garbage qubits, the original inputs $b_i$ and $c_i$. A single addition of three $n$-bit numbers requires a $O(n)$ 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 the following: $(u+v) = (a+b+c)[m]$, where $(u+v)$ is a CSE number.

In this section, we provide an alternative, pedagogical explanation of Gossett's modular reduction [8]. Later, we contribute a mapping to a 2D architecture, using unbounded fanout to maintain constant-depth for adding back modular residues. This last step is missing 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 $n$-bit, which we can truncate as long as we add back its modular residue $2^n \bmod m$. How can we guaran-

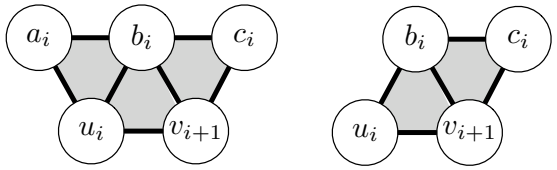


Figure 4: The carry-save adder (CSA), or 3-2 adder, and carry-save 2-2 adder.

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

Figure 2: An example of carry-save encoding for the 5-bit conventional number 30.

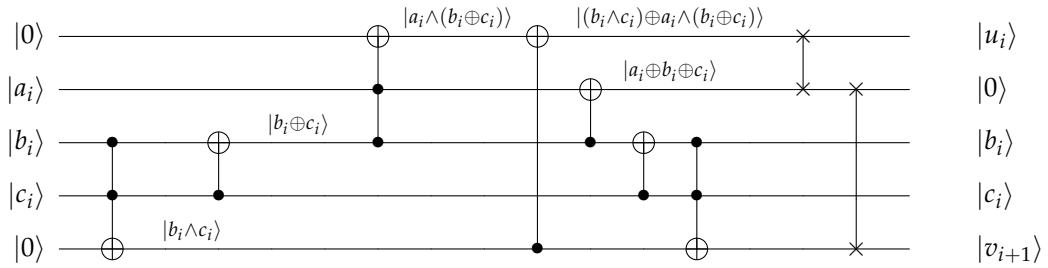


Figure 3: Carry-save adder circuit for a single bit position $i$: $a_i + b_i + c_i = u_i + v_{i+1}$.

tee 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 represents an $(n+1)$-bit conventional number and has $2n$ 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 get more overflow bits (of significance $2^{n-1}$ or higher). This equivalence is shown in Equation 6.

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

**Lemma 1** (Modular Reduction in Constant Depth [8])**.** *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 gen-

erated. This is illustrated for a 3-bit modulus, and 5-bit registers, in Figure 5.

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 at no layer is an overflow $(n+2)$-bit generated, 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 $u''_{(n,n+1)} + v''_{(n+1)} \leq 3 \cdot 2^{n+1}$, 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 implies 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. However, we omit this proof here for simplicity.

To summarize, the circuit resources for modular addition are $O(1)$ depth and $O(n)$ width.

## 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 6 which corresponds directly to the schematic proof in Figure 5. Figure 6 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 1) to distribute the control bit $v_4$ to its modular residue, which we denote as $|c^{v_4}\rangle \equiv |2^4[m] \cdot v_4\rangle$. $c^{v_4}$ 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 $u_4$: $|c^{u_4}\rangle \equiv |2^4[m] \cdot u_4\rangle$. $c^{u_4}$ 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.

| | | | | | | |
|---|---|---|---|---|---|---|
| | $a_4$ | $a_3$ | $a_2$ | $a_1$ | $a_0$ | 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 |

Figure 5: A schematic proof of Gossett's constant-depth modular reduction for $n = 3$

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 $u''' + v'''$.

# 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 numbers 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 6.2.

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 Numbers

We now consider the problem of multiplying a classical number controlled on a quantum bit and a *quantum number*,[2] which is a quantum superposi-

[2] In this paper, quantum numbers often result by entangling a classical number in one register with a quantum control bit. This should not be confused with the physics meaning of a quantum number.

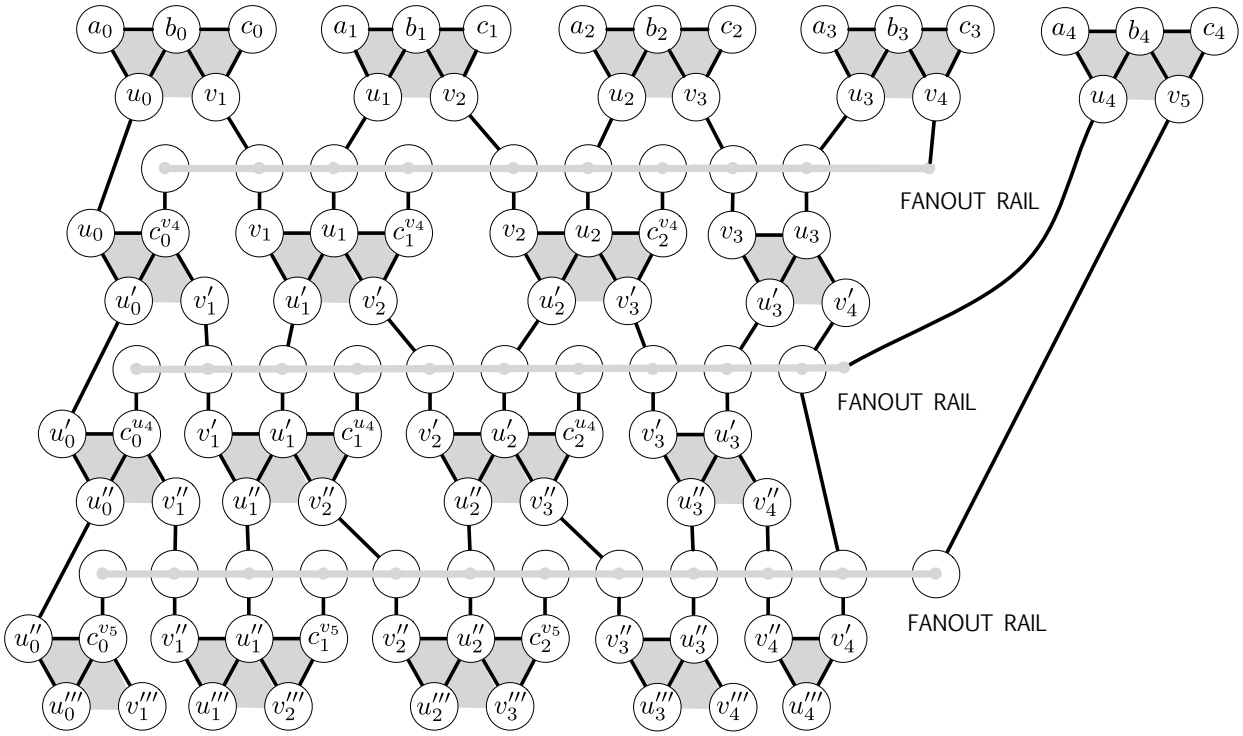


Figure 6: Addition and three rounds of modular reduction for a 3-bit modulus.

tion of classical numbers: given an $n$-qubit quantum number $|x\rangle$, 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 numbers are multiplied in series to a single quantum register.

We first create $n$ quantum numbers $\left|z^{(i)}\right\rangle$, which are shifted multiples of the classical number $a$ controlled on the bits of $x$: $\left|z^{(i)}\right\rangle \equiv \left|2^i a[m] \cdot x_i\right\rangle$. 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 this 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 copy all $n$ bits of $x$, $n$ times each, using an unbounded fanout operation so that $n$ copies of each bit $|x_i\rangle$ is 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 $x_i$. The schematic for this is shown in Figure 7, not showing how 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 6.2.

Finally, we tackle the most interesting problem: given two $n$-qubit quantum numbers $|x\rangle$ 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

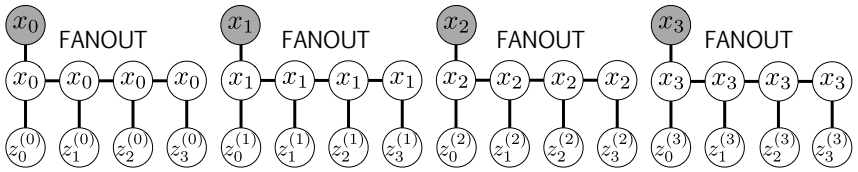


Figure 7: Creating $n = 4$ shifted values $\{z^{(0)}, z^{(1)}, z^{(2)}, z^{(3)}\}$ for an input number $x$.

exponentiation.

Instead of creating $n$ quantum numbers $\left|z^{(i)}\right\rangle$, we must create $n^2$ 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 numbers of $n$ qubits each takes depth $O(\log(n^2))$ which is still $O(\log n)$. Creating $n^2 \times n$-bit quantum numbers takes width $O(n^3)$.

## 6.2 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.

$$z^{(0)} + z^{(1)} + \ldots z^{(n-1)} \equiv (u + v)[m] \tag{7}$$

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. In the cases above, serial multiplication results in $t = n$ and parallel multiplication results in $t = n^2$. At the beginning of the circuit, all CSA tiles are *active* in that they have tile input numbers $z^{(i)}$ to multiply, and their tile outputs will affect the overall circuit output, $u + v$.

As the circuit proceeds through a number of timesteps, tiles will become *inactive* when they do not receive new numbers for their tile inputs; at that point, their tile outputs can no longer affect the circuit output. Since the CSA tile is a 3-2 adder, one can see that if there are $t$ CSA tiles active at the beginning of a timestep, there are $\lceil 2t/3 \rceil$ active tiles at the end of the timestep, since there are roughly two-thirds as many input numbers left to add down to the circuit output $u + v$. One can see that the total number of timesteps is therefore $\lceil \log_{3/2}(t/3) \rceil + 1$.

To facilitate the below discussion, we will assign colors to each CSA tile, which are updated during the circuit execution. Active tiles can either be black or gray. A *black* tile will keep its two output numbers as inputs and receive a third input number. An exception is the rightmost black tile may teleport one of its output numbers to its left black nearest neighbor and receive two input numbers from its right gray nearest neighbor. A *gray* tile will teleport one of its output numbers to the nearest active tile to its left and the other output number to the nearest active tile to its right. An exception is the rightmost gray tile may teleport both output numbers to its left black nearest neighbor. We can think of inactive tiles as *white* tiles in that they "fade" out of the circuit, and numbers get teleported through them without stopping to be added. The symbols for these colors are shown in Figure 8.

The rules for updating tiles at the end of each timestep are as follows:

- **Black tiles** are always active for the next timestep, but change colors as follows.
  - The leftmost tile always stays black.
  - If a black tile has a gray tile as its nearest active right neighbor in the current timestep, it stays **black** in the next timestep.
  - If a black tile has a black tile as its nearest active neighbor either to the right or the

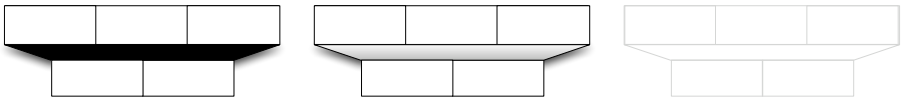

Figure 8: From left to right, the symbols for a black, gray, and white tile, respectively.

left, and it is not the leftmost tile, it turns **gray** in the next timestep.

- **Gray tiles** always turn white (inactive) in the next timestep.

The initial state of the tile colors depends on its index $i \in \{0, 1, \ldots, q-1\}$ within $q = \lceil t/3 \rceil$ tiles.

- If $i \bmod 3 = 0$, then it starts out black.
- If $i \bmod 3 = 1$, then it starts out gray.
- If $i \bmod 3 = 2$, then it starts out black.

Given the rules above, one can see that the leftmost tile stays black throughout the entire circuit, and holds the final output number $(u + v)$ at the end.

Each timestep of the circuit consists of the following operations:

1. All active CSA tiles will execute in parallel to transform their three input numbers into two output numbers (a CSE number).
2. Gray tiles teleport their output numbers to the left and to the right to their black tile neighbors. The exception is the rightmost gray tile will teleport both of its output numbers to its left black tile nearest neighbor.
3. Tile colors will change according to the rules above. Approximately two-thirds of the tiles will become inactive in the next timestep.
4. Go back to Step 1 for the next timestep.

These steps and the above tile color rules are best illustrated with a concrete example. In Figure 9, we see the circuit for modular multiple addition as a series of snapshots, separated by heavy dotted lines, with the passage of time going downward. The tiles change color over time, and the arrows indicate the teleportation of output numbers to neighboring active tiles in each timestep. In the initial timestep, the tiles are numbered to show how they are assigned their initial color. Between Timestep 0 and Timestep 1, all $\lceil n/3 \rceil$ CSA tiles are active. After each succeeding timestep, $\lfloor 2/3 \rfloor$ fewer CSA tiles are active until the very end, when only one CSA tile is active. By the convention established above, we teleport the rightmost output numbers to the left, so that the final output is read out from the leftmost CSA tile.

Now we can analyze the circuit resources for multiplying $n$-bit quantum numbers, 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 $O(n^3)$.

# 7 Quantum Modular Exponentiation

We now extend our arithmetic to modular exponentiation, which is repeated modular multiplication

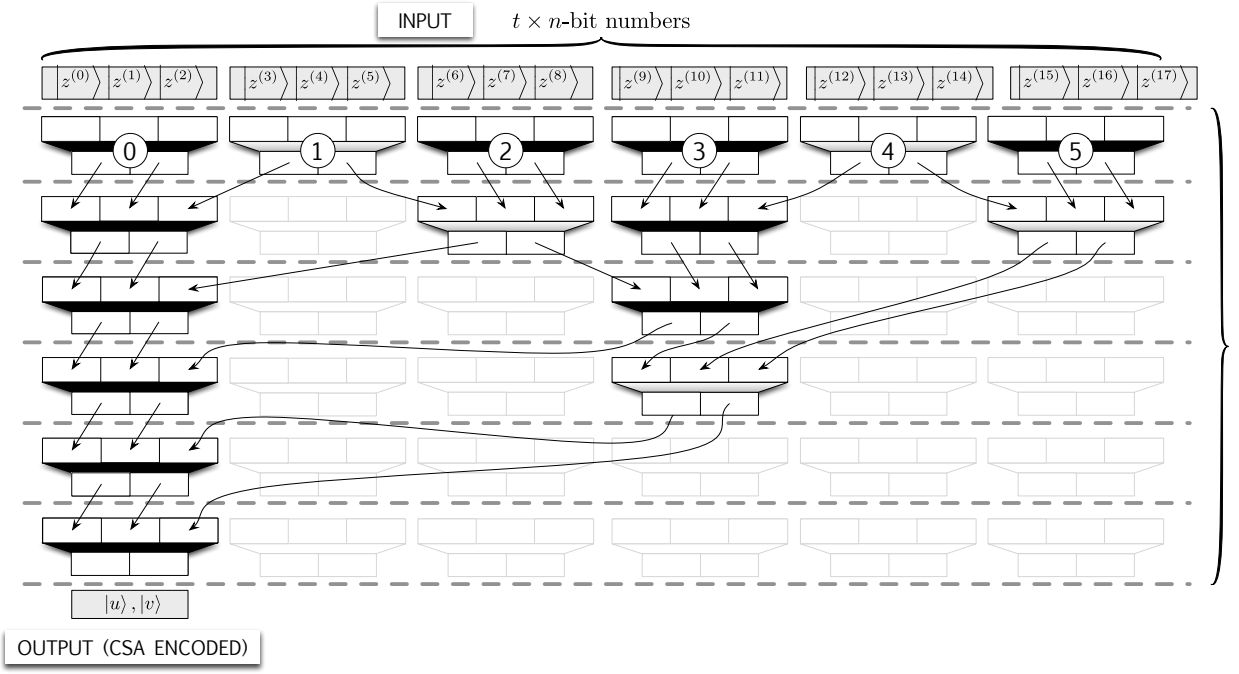


Figure 9: Modular multiple addition of quantum numbers on a CSA tile architecture for $t = 18$ with depth $(\lceil \log_{\frac{3}{2}}(t/3) \rceil + 1) = 6$ timesteps

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 numbers in series, since a classical number entangled with a quantum number is also quantum. It takes the same depth $O(t \log n)$ as the previous case.

Finally, we consider multiplying $t$ quantum numbers $\{x^{(0)}, x^{(1)}, \ldots, x^{(n-1)}\}$ in a parallel, logarithmic depth, binary tree. This is shown in Figure 10. 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 overall depth of this parallel modular exponentiation structure is $O(\log(t) \log(n))$. In phase estimation for QPF, it is sufficient to take $t = O(n)$ [12, 1]. Therefore our total depth is $O(\log^2(n))$ as desired. At this point, combined with the parallel phase estimation procedure of [1], 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 $O(n^4)$.

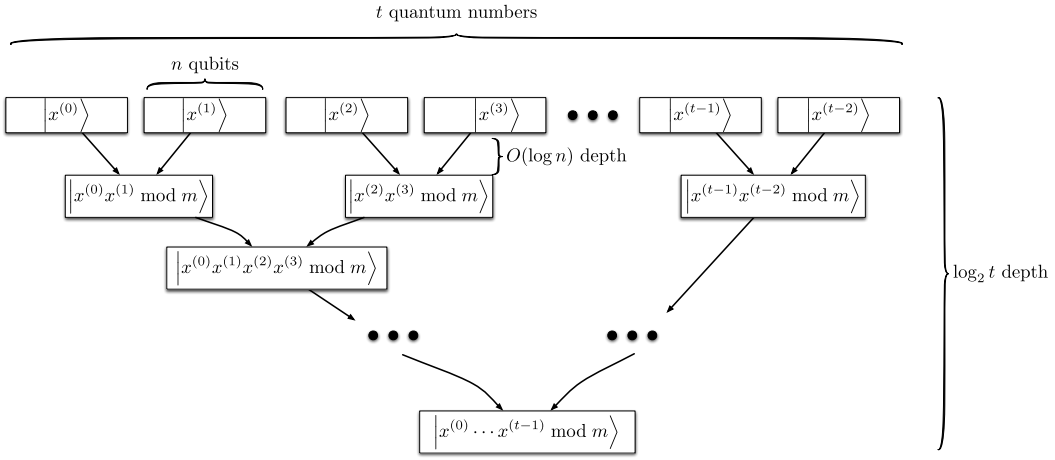


Figure 10: Parallel modular exponentiation: multiplying $t$ quantum numbers in a $O(\log(t)\log(n))$-depth binary tree.

| Implementation | Architecture | Depth | Width |
|---|---|---|---|
| Vedral, Barenco & Ekert [23] | AC | $O(n^3)$ | $O(n)$ |
| Gossett [8] | 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 [18] | AC | $O(n^3)$ | $O(n)$ |
| Fowler, Devitt, Hollenberg [7] | 1D NTC | $O(n^4)$ | $O(n^3)$ |
| Kutin [11] | 1D NTC | $O(n^2)$ | $O(n)$ |
| Current Work | 2D NTC | $O(\log^2 n)$ | $O(n^4)$ |

Table 1: Asymptotic resource usage for quantum factoring of an $n$-bit number.

# 8 Results

The resources required for our approach, as well as other nearest-neighbor approaches, are listed in Table 1, 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.

# 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 cen-

tral 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 [15]. It is an interesting open question how a generic conversion of a constant-depth CCAC factoring architecture [4, 10] 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.

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] M.N. Vyalyi A. Yu. Kitaev, A.H. Shen: *Classical and Quantum Computation*. American Mathematical Society, Providence, Rhode Island, 2002.

[2] Stephane Beauregard: Circuit for Shor's algorithm using 2n+3 qubits. May 2002. [arXiv:0205095].

[3] Anne Broadbent and Elham Kashefi: Parallelizing Quantum Circuits. April 2007. [arXiv:0704.1736].

[4] Dan E. Browne, Elham Kashefi, and Simon Perdrix: Computational depth complexity of measurement-based quantum computation. In *Proceedings of the Fifth Conference on Theory of Quantum Computation, Communication, and Cryptography (TQC)*, September 2010. [arXiv:0909.4673].

[5] Byung-Soo Choi and Rodney Van Meter: "$\Theta(\sqrt{n})$"-depth Quantum Adder on a 2D NTC Quantum Computer Architecture. August 2010. [arXiv:1008.5093].

[6] Thomas G. Draper: Addition on a Quantum Computer. August 2000. [arXiv:0008033].

[7] Austin G. Fowler, Simon J. Devitt, and Lloyd C. L. Hollenberg: Implementation of Shor's Algorithm on a Linear Nearest Neighbour Qubit Array. *Quantum Information and Computation*, 4:237–251, February 2004. [arXiv:0402196].

[8] Phil Gossett: Quantum Carry-Save Arithmetic. *Quantum*, 1998.

[9] Aram Harrow and Austin Fowler: Private communication, October 2011.

[10] Peter Høyer and Robert Špalek: Quantum Circuits with Unbounded Fan-out. *Theory of Computing*, 1(5):81–103, August 2005. [arXiv:0208043].

[11] Samuel A. Kutin: Shor's algorithm on a nearest-neighbor machine. August 2006. [arXiv:0609001].

[12] Michael A. Nielsen and Isaac L. Chuang: *Quantum Computation and Quantum Information*. Cambridge University Press, Cambridge, U.K., 2000.

[13] Paul Pham: Quantum compiling with parallelized phase estimation. Unpublished manuscript, February 2011.

[14] Robert Raussendorf, Daniel Browne, and Hans Briegel: Measurement-based quantum computation on cluster states. *Physical Review A*, 68(2), August 2003. [ doi:10.1103/PhysRevA.68.022312, arXiv:0301052 ].

[15] David Rosenbaum: Optimal Quantum Circuits for Nearest-Neighbor Architectures. April 2012. [arXiv:1205.0036].

[16] 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. [arXiv:9598927].

[17] Yasuhiro Takahashi and Noboru Kunihiro: A linear-size quantum circuit for addition with no ancillary qubits. *Quantum*, 5(6):440–448, 2005.

[18] Yasuhiro Takahashi and Noboru Kunihiro: A quantum circuit for Shor's factoring algorithm using 2n+2 qubits. *Quantum Information and Computation*, 6(2):184–192, 2006.

[19] Yasuhiro Takahashi, Seiichiro Tani, and Noboru Kunihiro: Quantum Addition Circuits and Unbounded Fan-Out. *Quantum Information and Computation*, 10(9–10):872–890, October 2010. [arXiv:0910.2530].

[20] Rodney Van Meter: *Architecture of a Quantum Multicomputer Optimized for Shor's Factoring Algorithm*. Ph.d., Keio University, 2006. [arXiv:0607065v1].

[21] Rodney Van Meter and Kohei Itoh: Fast quantum modular exponentiation. *Physical Review A*, 71(5), May 2005. [ doi:10.1103/PhysRevA.71.052320, arXiv:0408006 ].

[22] Rodney Van Meter, Kohei Itoh, and Thaddeus Ladd: Architecture-dependent execution time of Shor's algorithm. May 2005. [arXiv:0507023].

[23] Vlatko Vedral, Adriano Barenco, and Artur Ekert: Quantum networks for elementary arithmetic operations. *Physical Review A*, 54(1):147–153, July 1996. [ doi:10.1103/PhysRevA.54.147, arXiv:9511018 ].

[24] C. S. Wallace: A Suggestion for a Fast Multiplier. *IEEE Transactions on Electronic Computers*, EC-13(1):14–17, February 1964. [doi:10.1109/PGEC.1964.263830].

[25] Christof Zalka: Fast versions of Shor's quantum factoring algorithm. 1998. [arXiv:9806084v1].