HIGH PERFORMANCE QUANTUM MODULAR MULTIPLIERS

Rich Rines[†][*]and Isaac Chuang[†]

September 1, 2017

Abstract

We present a novel set of reversible modular multipliers applicable to quantum computing, derived from three classical techniques: 1) traditional integer division, 2) Montgomery residue arithmetic [1], and 3) Barrett reduction [2]. Each multiplier computes an exact result for all binary input values, while maintaining the asymptotic resource complexity of a single (non-modular) integer multiplier. We additionally conduct an empirical resource analysis of our designs in order to determine the total gate count and circuit depth of each fully constructed circuit, with inputs as large as 2048 bits. Our comparative analysis considers both circuit implementations which allow for arbitrary (controlled) rotation gates, as well as those restricted to a typical fault-tolerant gate set.

1 Introduction

1.1 Efficient Modular Multiplication Implementations

Circuits implementing reversible modular arithmetic operations are of significant contemporary interest due to their prevalence in important quantum algorithms [3]. Resource-efficient implementation of these operations is critical to their eventual implementation on a quantum computer. In this paper, we describe three novel techniques which asymptotically reduce the resources required for exact, reversible modular multiplication to those necessary for non-modular integer multiplication. Existing proposals for efficient reversible modular multipliers largely fall in one of two categories: (1) the composition of reversible modular adders, each comprising the equivalent of three reversible non-modular adders; or (2) approximated division, in which reversible modular reduction is simplified by allowing the return of an incorrect output for some subset of input values [4, 5]. Compared to those in the first category, our circuits achieve a factor-of-three reduction in asymptotic gate count and circuit depth while requiring only $\mathcal{O}(\log n)$ additional qubits. They perform comparably to those in the second category, but without employing arithmetical approximations–regardless of the impact of such approximations on the overall fidelity of quantum algorithms, our constructions demonstrate that accuracy need not be sacrificed in order to obtain an efficient modular multiplication circuit.

Classically, large-integer modular multiplication is a critical component of cryptographic functions such as the RSA public-key encryption system [6]. The standard strategy for modular multiplication is to first calculate the integer product of the input values, and then reduce the result via division. However, due to the inefficiency of integer division on most processors, systems implementing extensive modular arithmetic typically employ specialized methods to eliminate the division requirement of modular reduction. In particular, Montgomery residue arithmetic [1] and Barrett reduction [2] are commonly used techniques in applications requiring many reductions under the same modulus. In the Montgomery method, the problem is converted to an "N-residue" representation in which modulo-N reduction instead requires calculations under an easier modulus (e.g. a power of two). Barrett reduction takes advantage of

[*] Massachusetts Institute of Technology Lincoln Laboratory, Lexington, Massachusetts 02420, USA
[†] Massachusetts Institute of Technology, Cambridge, Massachusetts 02139, USA

Distribution Statement: A. Approved for public release - distribution is unlimited

This material is based upon work supported by the Assistant Secreta1ry of Defense for Research and Engineering under Air Force Contract No. FA8721-05-C-0002 and/or FA8702-15-D-0001. Any opinions, findings, conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Assistant Secretary of Defense for Research and Engineering.

the constant modulus by pre-computing a fixed-point reduction factor once, so that individual reductions can be computed using only multiplication and bit-shift operations.

In this work, we describe three novel reversible modular multipliers, employing (1) a new implementation of the standard division-based reduction procedure, as well efficient reversible adaptations of (2) classical Montgomery multiplication and (3) Barrett reduction. Our designs principally comprise common abstract circuit primitives, such as reversible adders and subtracters, enabling their implementation within various architectural and arithmetical models. In particular, the Montgomery and Barrett multipliers introduced are uniquely amenable to arithmetic in the quantum Fourier transform basis, sidestepping the bottlenecks plaguing many previous implementations utilizing this representation [7, 8]. We discuss the relative trade-offs of various implementation methods for each modular multiplication strategy, and perform a detailed resource analysis of each.

1.2 Prior Modular Multiplication Implementations

The first quantum circuits for modular multiplication were developed as part of a typical arithmetical hierarchy [9, 10, 11],

$$\text{(Integer Adder)} \longrightarrow \text{(Modular Adder)} \longrightarrow \text{(Modular Multiplier)},$$

in which a quantum modular multiplier is constructed from a sequence of modular adders, which in turn consist of multiple integer addition steps. The complexity of modular multiplication is then driven by the complexity of reversible implementations of integer addition.

The first quantum integer addition circuits were based on the classical ripple-carry adder, which has circuit depth that is linear in the bit-width of its addends [10, 12]. The circuit depth and total size of modular multiplication using the ripple-carry adder is $\mathcal{O}(n^2)$ for n-bit inputs. Logarithmic-depth addition circuits were subsequently proposed [13] that reduce the depth of modular multiplication to $\mathcal{O}(n \log_2 n)$, with the total number of gates remaining $\mathcal{O}(n^2)$.

Quantum adders that operate in the quantum Fourier transform basis [7] have also been used to implement modular addition, multiplication, and exponentiation. By moving integer addition to the Fourier basis, Beauregard [8] presented a modular multiplication circuit that has depth $\mathcal{O}(n^2)$ and total circuit count $\mathcal{O}(n^3)$, but requires fewer qubits than previously described circuits. However, Fourier-basis arithmetic employs arbitrarily angled rotation gates, which may require more resources to implement on typical architectures than the reversible AND (i.e. TOFFOLI) gates required by the other adders. For this reason, it is difficult to do a direct comparison of Fourier-basis and binary circuits.

Using Fourier-basis addition, Kutin has devised a unique and efficient procedure for approximate quantum modular multiplication with linear algorithmic depth and a linear number of qubits [5]. This circuit uses an approximate multiplication technique introduced by Zalka [4], which requires uncertain assumptions regarding the distribution of systematic errors. The argument is made that these errors do not lead to an appreciable level of error.

An exact, linear-width quantum modular multiplication procedure was proposed by Pavlidis and Gizopoulos [14]. By applying Fourier-basis arithmetic to construct exact reversible constructions of a large array of arithmetic operations, they introduce a novel quantum Granlund-Montgomery division procedure enabling exact quantum modular multiplication with linear depth and linear width. Still, the leading terms in this construction remain prohibitive, requiring $9n$ qubits, $800n^2$ total quantum gates, and a parallelized depth of $1000n$ rotation gates, with almost 90% of these costs devoted to the division procedure.

As in classical systems [15], a trade-off exists between algorithmic depth and width. Allowing a super-linear supply of ancillary qubits, various procedures for sub-linear time quantum multiplication have been introduced. Gossett's carry-save multiplier achieves logarithmic depth but requires $\mathcal{O}(n^2)$ qubits [16]. Similarly, modular exponentiation can be performed by arranging the component multipliers in a logarithmic-depth binary-tree structure at the cost of a quadratic circuit width [17]. Combining these models with constant-depth teleportation-based fan-out, Pham and Svore have introduced a 2D nearest-neighbor carry-save architecture allowing modular exponentiation in $\mathcal{O}(\log^2(n))$ depth with $\mathcal{O}(n^4)$ qubits [18].

Classically, multipliers exploiting the fast Fourier transform (FFT) and convolution theorem have long dominated in asymptotic depth [19, 20, 21]. Accordingly, the fastest known quantum multipliers are direct applications of these constructions [4, 22]. However, a signature of both classical and quantum convolution-based multipliers is a prohibitive initial cost–for example, Zalka's quantum Schönhage-Strassen multiplier [4] has an asymptotic depth of $\sim 2^{16}n^{0.2}$.

We summarize these results in Table (1). Because of the difficulty of making side-by-side comparisons of Fourier-basis and binary circuits, we have separated the two and characterized each in terms of their principle operation (Toffoli gates in the case of binary arithmetic, total controlled-rotations in the Fourier case.) The characteristics for binary modular multipliers utilizing modular adders are determined from the particular adders employed, assuming three adds per modular addition and $2n$ modular adds per modular multiplication. All other circuits are presented as reported in the reference. We will discuss the various trade-offs that exist, and where our proposed circuits fall among these, in Section 7.

Table 1: Resource comparison of in-place quantum modular multipliers. Only the leading order term is shown for each count.

Binary Arithmetic:

Proposal	Architecture	Qubits	Gates†	Depth†
*Cuccaro et. al. [12]	Modular Addition (Ripple-Carry)	$3n$	$12n^2$	$12n^2$
*Draper et. al. [13]	Modular Addition (Prefix)	$5n$	$60n^2$	$24n\log_2 n$
Zalka [4]	Schönhage-Strassen (FFT)	$24...96n$	$2^{16}n$	$2^{16}n^{0.2}$
Pham-Svore [18]	Carry-Save (nearest-neighbor)	$16n^2$	$384n^2\log_2 n$	$56\log_2 n$
This work	**Exact Division (Prefix)**	$\boldsymbol{5n}$	$\boldsymbol{20n^2}$	$\boldsymbol{8n\log_2 n}$
	Montgomery Reduction (Prefix)	$\boldsymbol{5n}$	$\boldsymbol{20n^2}$	$\boldsymbol{8n\log_2 n}$
	Barrett Reduction (Prefix)	$\boldsymbol{5n}$	$\boldsymbol{20n^2}$	$\boldsymbol{8n\log_2 n}$
	Exact Division (Ripple)	$\boldsymbol{3n}$	$\boldsymbol{4n^2}$	$\boldsymbol{4n^2}$
	Montgomery Reduction (Ripple)	$\boldsymbol{3n}$	$\boldsymbol{4n^2}$	$\boldsymbol{4n^2}$
	Barrett Reduction (Ripple)	$\boldsymbol{3n}$	$\boldsymbol{4n^2}$	$\boldsymbol{4n^2}$

*Reference proposes an adder only. We assume 3 adders per modular add, $2n$ modular adds per multiply.
†Total gate counts and depths provided in Toffoli gates.

Table 2: Resource comparison of Fourier-basis in-place quantum modular multipliers. Gate counts are reported in two ways: (1) the total number of rotations assuming infinite-precision control, and (2) the number of gates after removing those with exponentially-small rotation angles [23]. Only the leading order term is shown for each count.

Fourier-basis arithmetic:

Proposal	Architecture	Qubits	Gates (1)‡	Gates (2)‡	Depth‡
Beauregard [8]	Modular Addition	$2n$	$4n^2$	$\mathcal{O}(n^3\log n)$	$8n^2$
Kutin [5]	Approximate Division	$3n$	$3n^2$	$2n^2$	$6n$
	Approximate (nearest-neighbor)	$3n$	$5n^2$	$4n^2$	$11n$
Pavlidis [14]	Granlund-Montgomery Division	$9n$	$800n^2$	$\sim 250^2$	$1000n$
This work	**Exact Division**	$\boldsymbol{2n}$	$\boldsymbol{2n^2\log_2 n}$	$\boldsymbol{2n^2}$	$\boldsymbol{8n\log_2 n}$
	Montgomery Reduction	$\boldsymbol{2n}$	$\boldsymbol{5n^2}$	$\boldsymbol{2n^2}$	$\boldsymbol{14n}$
	Barrett Reduction	$\boldsymbol{2n}$	$\boldsymbol{5n^2}$	$\boldsymbol{2n^2}$	$\boldsymbol{14n}$

‡Total gate counts and depths in arbitrarily-angled controlled-rotations.

1.3 Outline of the Paper

The remainder of the paper is organized into 6 sections and one appendix. In the next section (Section 2) we provide background required by all the circuit constructions described in this paper. We also use these constructions to describe the most common implementation of modular multiplication, building up the multiplier as a sequence of modular adders. In Section 3 we describe the first of our modular multiplication circuits; a circuit that uses a new implementation of the standard division technique. In Section 4 we describe a circuit that uses the Montgomery reduction technique and in Section 5 we describe the implementation of a modular multiplier based on Barrett reduction. In Section 6 we describe details of the implementation of our schemes that are specific to Fourier basis arithmetic. In Section 7 we present a resource analysis of each of the new implementations that compares the resources required by our circuits to those required by the base implementation constructed from modular adders. We perform the resource analysis for circuits utilizing a variety of adders including: carry-ripple, carry look-ahead and Fourier basis. Finally in the appendix we describe details related to the integer adders and multiplier circuits used in the constructions, including several new implementations.

2 Quantum Modular Multiplication

In this section we describe the basic methods and circuits that are used to construct quantum modular multipliers. We also describe the standard circuit that is most often used for quantum modular multiplication, which performs the modular multiplication using modular adders. This circuit will provide the baseline of comparison for our new methods.

Most quantum algorithms require "quantum-classical" modular multipliers, in which one multiplier input is a fixed classical parameter. The circuits introduced in this paper are therefore described in this form. We will also describe the modifications required to implement full "quantum-quantum" modular multipliers. These circuits would be useful, for example, in the quantum circuit for elliptic curves [24].

In the quantum-classical case, our goal is described by the quantum operation,

$$\left|y\right\rangle_n \longrightarrow \left|Xy \bmod N\right\rangle_n, \tag{1}$$

where X is a classically-determined multiplicand, $n \triangleq \lceil \log_2 N \rceil$ is the number of bits required to hold the modulus N, and we have used subscripts to indicate the size of the requisite quantum registers (ignoring any intermediate ancilla qubits). In the quantum-quantum case, x is held in an additional quantum register, which is preserved through the operation:

$$\left|x\right\rangle_n \left|y\right\rangle_n \longrightarrow \left|xy \bmod N\right\rangle_n \left|y\right\rangle_n. \tag{2}$$

In general, we will reserve uppercase variable names for constant classical parameters.

2.1 Modular Multiplication

A non-modular quantum multiplier can be constructed using the techniques of grade-school arithmetic: we compute and accumulate a sequence of partial products. In binary arithmetic, each partial product is computed by multiplying a single bit of the multiplier y by the shifted multiplicand $2^k X$, so that the product yX is computed by accumulating $\sum_k y_k(2^k X)$ for each bit y_k of y. A binary quantum multiplier therefore consists of a sequence of quantum additions, controlled by the bits of a quantum input $\left|y\right\rangle$.

A modular multiplier can be constructed by combining the steps of non-modular multiplication and integer division operators, where the modular product is returned as a remainder from the latter. In general, an information-preserving divider will compute both quotient and remainder terms. For reversible modular modular reduction, we therefore must take additional steps to uncompute the quotient. Applying Bennett's technique for constructing reversible circuits from classical operations [25], in which we pseudo-copy the result into an ancilla register and reverse the entire computation, we implement,

$$\begin{aligned}
\left|0\right\rangle \left|y\right\rangle \left|0\right\rangle &\longrightarrow \qquad \left|Xy\right\rangle \left|y\right\rangle \left|0\right\rangle \\
&\longrightarrow \left|q\right\rangle \left|Xy - qN\right\rangle \left|y\right\rangle \left|0\right\rangle \\
&\longrightarrow \left|q\right\rangle \left|Xy - qN\right\rangle \left|y\right\rangle \left|Xy - qN\right\rangle \\
&\longrightarrow \qquad \left|Xy\right\rangle \left|y\right\rangle \left|Xy - qN\right\rangle \\
&\longrightarrow \qquad \left|0\right\rangle \left|y\right\rangle \left|Xy - qN\right\rangle,
\end{aligned} \tag{3}$$

where $\left|Xy - qN\right\rangle = \left|Xy \bmod N\right\rangle$. Unfortunately, the above computation is not very resource efficient. In addition to requiring additional ancilla registers to store results during the uncomputation, both the multiplication and division components of the modular multiplier must be implemented twice: once to compute the result, and again in reverse to uncompute and clear the ancilla space.

Due to the complexity of reversible division-based modular reduction, early modular multipliers were instead constructed with reductions embedded within the multiplication, one after each addition of a partial product [9, 10, 8, 17]. If we classically reduce each partial product modulo N prior to its addition, then at most one value of N needs to be subtracted with each step. We can therefore equivalently implement reversible circuits for modular addition, so that no garbage is left after any step of the multiplier. The disadvantage of this approach is that each reversible modular addition requires

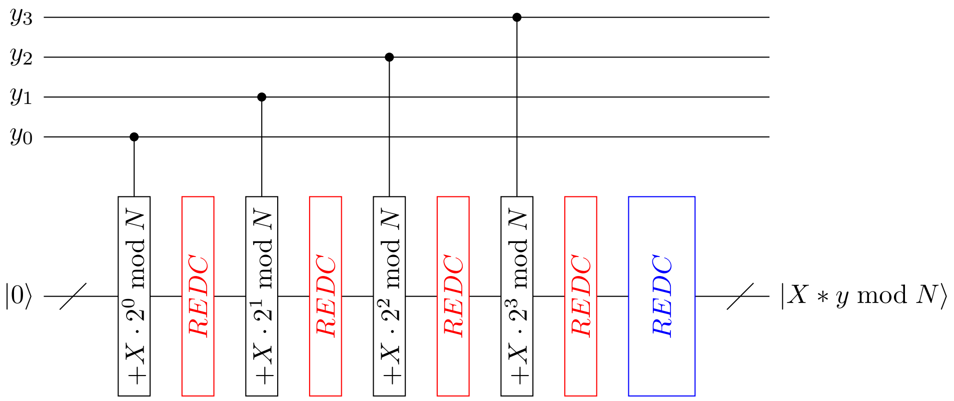


Figure 1: A 4-bit quantum modular multiplier that multiplies a quantum register by a classical constant. We either reduce modulo N after every addition (red blocks), or once for the entire multiplier (blue block). In both cases the multiplier consists of n conditional additions of n-bit constants.

multiple integer adders, increasing the overhead of the modular multiplier linearly by this factor. In Figure 1 we illustrate the two methods for modular multiplication: using modular adders, or using an integer multiplication followed by a division step.

The circuits introduced in this paper derive from the first, division-style reduction strategy. We are able to construct modular multipliers with a single integer multiplication step and a division step whose size and depth is logarithmic in the size of the inputs. While quantum circuits for modular multiplication have previously been developed employing this reduction style [4, 5, 14], the efficient reversible implementation of the modular reduction stage in this construction has remained a bottleneck. In the case of [4, 5], approximations are used that deterministically introduce errors for some inputs, which are argued to have an insignificant impact on the error rate of the circuit. In [14], a precise division-style reduction circuit is introduced. However, the overhead of their circuit is similar to the compute, copy, uncompute approach described above.

2.2 In-place Modular Multiplication

The operation shown in Equation (1) is an example of an *in-place* operation, where one of the input registers is overwritten by the output. To construct an in-place modular multiplier we can use the standard method that utilizes the reverse of the inverse function. For example, the out-of-place quantum-classical modular multiplier performs the transformation: $|y\rangle|0\rangle \longrightarrow |y\rangle|X*y \bmod N\rangle$. The inverse function with the modular product as input performs the transformation: $|X*y \bmod N\rangle|0\rangle \longrightarrow |X*y \bmod N\rangle|y\rangle$. Both these functions produce the same output and therefore we can chain them as:

$$|y\rangle\,|0\rangle \xrightarrow{(*X)_{fwd}} |y\rangle\,|Xy \bmod N\rangle \xrightarrow{\text{SWAP}} |Xy \bmod N\rangle\,|y\rangle \xrightarrow{(*X^{-1})_{rev}} |Xy \bmod N\rangle\,|0\rangle\,. \tag{4}$$

For the quantum-classical multiplier the inverse of the modular multiplication by X is a modular multiplication by $X^{-1} \bmod N$. Since X is classical this inverse can be calculated offline using the extended Euclidean algorithm. The in-place modular multiplier is then just two out-of-place multipliers.

2.3 Controlled Modular Multiplication

For most applications of the modular multiplier we will need to condition its operation on an external control qubit. There are three main ways that this can be done. The first way is to add the external control to all gates in the circuit. This adds significant overhead to the circuit and require every gate to share a common control qubit, either sequentializing the circuit or forcing a pseudo-copy of the control to a set of ancillas to allow gates to execute in parallel.

The second technique is to incorporate the global control into the individual controls already required by each adder in the multiplier. For each addition we can use a Toffoli gate to combine the external control with its local control in the input register (i.e., bit y_i from Figure 1) onto an ancilla bit, and then use this bit to control the adder.

A related and third way to control arbitrary quantum operations is to shift the input state into a subspace of the overall computational Hilbert space that is returned to its initial state after the operation. This methodology generally requires an additional qubit register serving as an auxiliary "quantum cache" [26]; however, in the case of in-place quantum multiplication, we can bypass this requirement at the cost of $\mathcal{O}(n)$ additional Fredkin [27] (controlled-swap) gates. For the out-of-place multiplier, if the quantum input register $|y\rangle$ is zero, we expect the output of the multiplication by it to be zero regardless of X. The multiplications by X and X^{-1} will therefore have the same output. Further, because we implement multiplication using controlled additions into an ancilla register, if $|y\rangle{=}|0\rangle$ the value in the accumulation register is left unchanged. We can therefore swap the input into the accumulation register, so that we compute,

$$|y\rangle\,|0\rangle \xrightarrow{\text{SWAP}} |0\rangle\,|y\rangle \xrightarrow{*X} |0\rangle\,|y + 0 * X\rangle = |0\rangle\,|y\rangle\,. \tag{5}$$

The fact that the additions are done modulo N does not matter because, for $y < N$, no reductions are ever required. If we then lift the subsequent swap step of Equation (4), the two out-of-place multiplications will cancel one another and $|y\rangle$ will be returned at the end of the operation:

$$|y\rangle\,|0\rangle \xrightarrow{\text{SWAP}} |0\rangle\,|y\rangle \xrightarrow{(*X)_{fwd}} |0\rangle\,|y\rangle \xrightarrow{(no\ \text{SWAP})} |0\rangle\,|y\rangle \xrightarrow{(*X^{-1})_{rev}} |0\rangle\,|y\rangle \xrightarrow{\text{SWAP}} |y\rangle\,|0\rangle\,. \tag{6}$$

In total, we require three sets of Fredkin gates for this implementation of the controlled multiplier: two negatively-controlled sets to swap the input register into and out of the "cache" register, and an additional positively-controlled set to control the swap at the center of Equation (4). Each can be implemented with one Toffoli gate and two CNOT gates per bit.

2.4 Modular Addition

There is an established method for constructing a reversible modular adder out of reversible integer adders. The method that we describe is a slight modification to those described in [9][11]. This method requires 3 in-place integer adders for an in-place modular quantum-classical or quantum-quantum addition. A circuit for this adder is shown in Figure 2. The circuit is for a quantum-classical modular adder. To make this a full quantum-quantum adder we would replace the additions involving X with full quantum integer additions. We have broken each integer adder into forward and reverse sections. An in-place adder requires both sections; therefore the modular adder is comparable in complexity to three in-place adders. Note, this adder is controlled by a single external qubit $|p\rangle$ as would be required if we use it as part of a multiplier.

Both input values to the modular adder are assumed to be reduced, i.e. $< N$. The modular adder works by first adding X in-place to the second quantum register containing y. We then subtract N out-of-place. If $X + y > N$, indicating that a reduction is necessary, then the result of this subtraction will be positive, otherwise it is negative. For two's complement addition a negative result will have the most-significant-bit (msb) set and this can be used as a control to the reverse segment of the subtraction to either overwrite the input or reverse the action of the forward subtraction. However, we must copy the msb in order to control the reverse subtraction and this produces a bit of garbage. Other than this bit of garbage, we have the result of the modular addition in the y register. To clear the garbage bit, we subtract X from the result and note that if we performed the reduction in the second step we now have $X + y - N - X = y - N < 0$ in the register and if we did not perform the reduction we have $X + y - X = y > 0$ in the register. Therefore we can use the sign of this subtraction to clear the garbage bit. We then uncompute the subtraction of X to complete the modular addition.

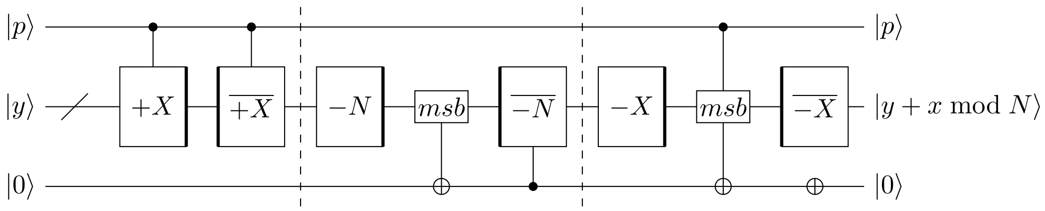


Figure 2: A controlled Modular quantum-classical adder constructed from 3 integer adders. Adders with thick lines on the right side indicate forward out-of-place additions and adders with thick lines on the left side are the corresponding reverse functions. A pair of these two comprise a full in-place adder. The *msb* function extracts the most significant bit from the register.

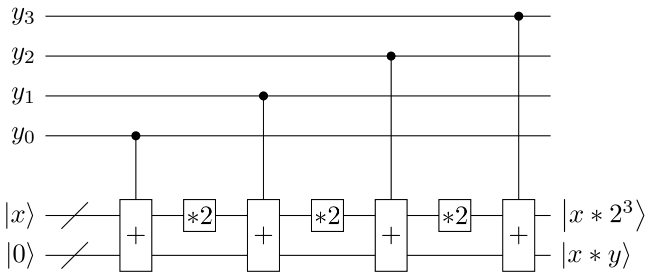


Figure 3: A 4-bit quantum multiplier constructed from a sequence of controlled additions

2.5 Quantum-Quantum Modular Multiplication

A full quantum-quantum multiplier (Figure 3) computes the product $|xy\rangle$ from to quantum input values $|x\rangle$ and $|y\rangle$. As in the case of the quantum-classical multiplier, the quantum-quantum multiplier consists of a sequence of controlled additions into an accumulation product register. If we use modular adders in the quantum-quantum multipliers then the resulting product will be reduced modulo N. Otherwise an additional reduction step will be required.

As discussed previously, each reduced partial product is of the form $y_i(2^i x) \bmod N$. For the quantum-classical modular multiplier we can calculate $2^i X \bmod N$ off-line; however, for the full quantum-quantum multiplier $|x\rangle$ is a quantum value and therefore $2^i x \bmod N$ must be calculated with a quantum circuit. Each addition in the multiplier uses a value that is twice the previous value, therefore we just need to shift the value by one position for each addition. A circuit that performs a shift and reduce is shown in Figure 4. The circuit shifts the input forward by one position by relabeling the input and inserting a zero at position 0. Since this produces a value $< 2N$ at most one reduction is required. We can perform this reduction by subtracting N and checking to see if the result is negative. If it is, we reverse the subtraction, otherwise we complete the in-place subtraction. We can clear the comparison bit by noting that since N is always chosen to be odd, and the pre-reduced shifted value is even, the output value is only odd when we have done a reduction. Therefore, the least significant bit can be use to clear the comparison bit.

For the quantum-classical modular multiplier, we could classically pre-compute the modular inverse $X^{-1} \bmod N$ required for the reversed step. In the quantum-quantum case, we instead require the inverse of a quantum value, requiring a reversible modular inversion routine. Reversible inversion circuits employing the extended Euclidean algorithm have been demonstrated [?]; however, their cost is much higher than that of a single modular multiplication, and incorporating them is beyond the scope of this paper. We will therefore describe and determine resources for only the out-of-place implementation of quantum-quantum modular multipliers.

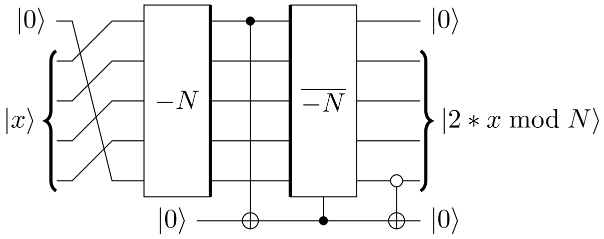


Figure 4: Modular shift and reduce circuit. Shown here for a 4-qubit register. The input value is shifted one position by relabeling the inputs and inserting a zero value at the LSB. At most one reduction of N is required, and after this reduction the most significant bit is cleared, which replaces the input ancilla used at the LSB.

3 Quantum Modular Multiplication with Division

Our first implementation of a quantum modular multiplier is the most straightforward: after an initial multiplication, we implement a reversible division operation comprising trial subtractions and controlled re-additions. Standalone modular reduction being irreversible, we first define a quantum division operator,

$$|t\rangle_{n+m} \xrightarrow{\text{Q-DIV}(N)} |t \text{ div } N\rangle_m \, |t \bmod N\rangle_n = |q\rangle_m \, |t - qN\rangle_n \,, \tag{7}$$

where $|t\rangle_{n+m}$ is the $(n+m)$-bit result of the initial multiplication, and (div) indicates integer division such that $(t \text{ div } N) = \lfloor t/N \rfloor = q$ is the computed quotient.

Classically, modular reduction is constructed from a division operation by simply discarding the quotient and preserving the remainder. In the reversible case, the quotient must instead be uncomputed, exacerbating the computational discrepancy between multiplication and division and sourcing the principal bottleneck in typical quantum implementations of modular multiplication. Here, we utilize information present in both the input and output registers of the out-of-place modular multiplier in order to clear $|q\rangle_m$ while circumventing a full Bennett-style reverse computation. The depth of the Q-DIV operator is then poly-logarithmic in n, so that the out-of-place modular multiplication operation,

$$|0\rangle_{n+m} \, |x\rangle_n \longrightarrow |t\rangle_{n+m} \, |x\rangle_n \xrightarrow{\text{Q-DIV}(N)} |q\rangle_m \, |t \bmod N\rangle_n \, |x\rangle_n \longrightarrow |0\rangle_m \, |t \bmod N\rangle_n \, |x\rangle_n \tag{8}$$

is asymptotically dominated by the initial computation of $|t\rangle$.

3.1 Multiplication Stage

We first must compute a state $|t\rangle$ such that t is congruent to the non-modular product $X * y$. For the purpose of modulo-N multiplication, we can reduce partial products prior to their accumulation, replacing (Xy) with,

$$t \triangleq \sum_{k=0}^{n-1} y_k \big(2^k X \bmod N\big), \tag{9}$$

such that $t \equiv Xy \pmod N$ and is bound by $t < nN$. We then require at most $n + m = \lceil \log_2(Nn) \rceil = n + \lceil \log_2 n \rceil$ bits to hold $|t\rangle_{n+m}$, so that the initial multiplication stage,

$$|0\rangle_{n+m} \, |y\rangle_n \xrightarrow{\text{Q-MAC}(X|N)} |t\rangle_{n+m} \, |y\rangle_n \,, \tag{10}$$

consists of n in-place, width-$(n+m)$ quantum adders conditioned on the bits of $|y\rangle_n$.

3.2 Division Stage

Using $|t\rangle_{n+m}$ as the input to the quantum division operation, we require at most $m \triangleq \lceil \log_2 n \rceil$ bits for the quotient register $|q\rangle_m$. We compute the quotient bitwise, beginning with its MSB, q_{m-1}. After unconditionally subtracting $2^{m-1}N$, the sign bit (MSB) of the register indicates whether the input $t < 2^{m-1}N$. Using the sign to condition the in-place re-addition of $2^{m-1}N$ onto the remaining bits of the register, we return its modulo-$(2^{m-1}N)$ reduction. We are left with the single sign bit indicating that the subtraction was undone, that is, that $q_{m-1} = 0$. After inverting the sign, we have therefore performed the operation,

$$|t\rangle_{n+m} \longrightarrow |q_{m-1}\rangle \left|t - q_{m-1}2^{m-1}N\right\rangle_{n+m-1} = |q_{m-1}\rangle \left|t \bmod (2^{m-1}N)\right\rangle_{n+m-1}, \tag{11}$$

where the resulting value in the accumulator is bound by $2^{m-1}N$, requiring only the $(n+m-1)$ LSBs of the register and allowing q_{m-1} to be preserved in the MSB location.

We iterate this process for $k = m-1, ..., 0$, each time reducing the register modulo-2^kN and computing the bit q_k of $|q\rangle_m$. After the final $(k = 0)$ reduction, we have constructed $|q\rangle_m$ over the m MSBs of the input register, while leaving the remainder $|t \bmod N\rangle_n$ in the remaining n bits. The complete Q-DIV(N) operation is shown as a quantum circuit in Figure 5.

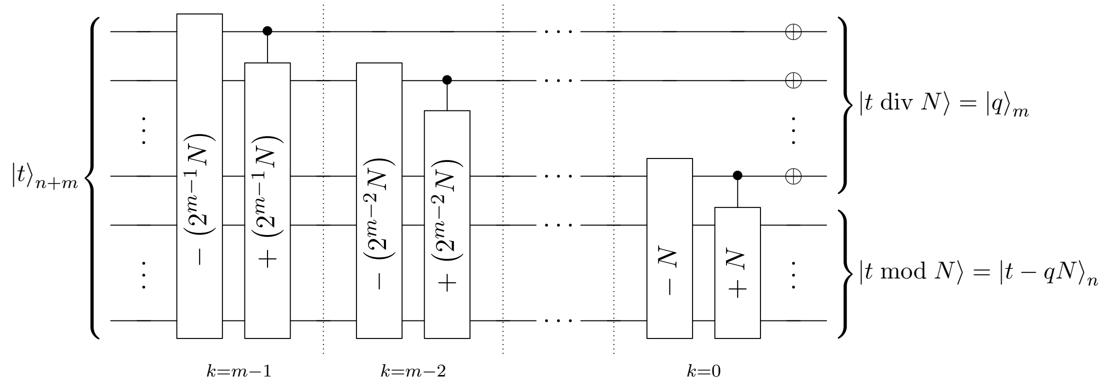


Figure 5: Quantum division operation described in Section 3.2. At each step k, we perform a trial subtraction and conditional re-addition of 2^kN, computing one (inverted) bit of the quotient q while reducing the input state modulo-(2^kN). The subtraction and re-addition of each stage can be merged into a single in-place quantum select-undo adder. (see the Appendix)

As described in the Appendix, an in-place quantum adder generally comprises a pair of out-of-place adders. In the select-undo quantum adder construction introduced in Appendix A.2, control is efficiently added to the in-place adder through the second adder in the pair, which selectively undoes the addition performed by the primary adder. In this structure, we require a control qubit only for the second adder. We can use the select-undo adder to merge the trial subtractions and subsequent conditional re-additions of the division circuit, performing the trial subtractions out-of-place and using the sign (MSB) of the difference to conditionally undo the result. As each subtraction by 2^kN affects only the bits more significant than k, the division operator comprises in total m in-place, n-qubit select-undo adders. Assuming logarithmic-depth prefix adders (Appendix A.1), the overall depth of this stage is $\mathcal{O}(m \log n) = \mathcal{O}(\log^2 n)$, with $\mathcal{O}(n \log n)$ total quantum gates.

Notably, the division stage constructed here assumes nothing of the initial multiplicands used to construct t, taking only the state $|t\rangle$ and the classical modulus as inputs. This stage is therefore independent of whether we are implementing a quantum-classical or quantum-quantum modular multiplier.

3.3 Uncomputation Stage

We now must uncompute the quotient register. Unlike the division stage, this requires the incorporation of the individual multiplicands or partial products used in the calculation of $|t\rangle$. However, we can avoid the complete reversal of the steps computing $|q\rangle_m$ by utilizing information contained in both the input $|y\rangle_n$ and output $|t \bmod N\rangle_n$ registers.

Our strategy requires that we first multiply the quotient register by the modulus in-place:

$$|q\rangle_m \xrightarrow{\text{Q-MUL}(N)} |qN\rangle_m , \tag{12}$$

where modulo-2^m reduction is implicit in the size of the m-bit register. As described in Appendix A.3, for odd N we can reversibly multiply $q * N \pmod{2^m}$ in-place, with $m-1$ quantum adders of sizes $1, ..., m-1$.

We then add the m LSBs of $|t \bmod N\rangle$ to the quotient register,

$$|qN\rangle_m \longrightarrow |qN + (t \bmod N)\rangle_m = |qN + (t - qN)\rangle_m = |t\rangle_m , \tag{13}$$

leaving us with the result computed in the initial multiplication stage, truncated to its m LSBs. We can clear $|t\rangle$ by a reverse of the multiplication stage, truncating all operations to their first m bits.

Though we now require only m-bit addition, our use of reduced partial products in computing $|t\rangle$ requires that we perform n total additions each controlled by a bit of $|y\rangle_n$. Given logarithmic-depth quantum adders, the depth of this stage would then be $\mathcal{O}(n \log\log n)$, dominating the $\mathcal{O}(\log^2 n)$ depth of the Q-DIV operation. We therefore make use of the work bits necessary for the multiplication and division stage adders to parallelize the narrower adders required of the uncomputation stage.

Dividing the $\mathcal{O}(n)$ work bits into $\mathcal{O}(n/m) = \mathcal{O}(n/\log n)$ separate accumulation registers, we can distribute and sum the n addends in groups of $\mathcal{O}(m)$. The independent accumulators can then be combined with $\mathcal{O}(\log(n/m)) = \mathcal{O}(m)$ quantum-quantum adders in a binary tree structure in order to compute the complete sum $|t\rangle_m$. After using the result to clear the quotient register, we must uncompute the individual accumulators by reversing the parallelized adds. The overall depth of this procedure is then $\mathcal{O}(m \log m) = \mathcal{O}(\log n \log\log n)$, no longer dominating the $\mathcal{O}(\log_2^2 n)$-depth division stage.

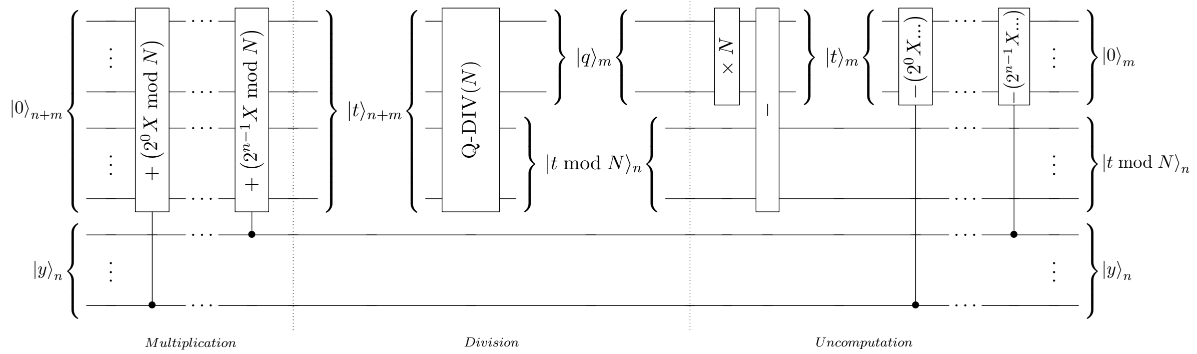


Figure 6: Out-of-place modular multiplier constructed from the Q-DIV(N) operation. The final sequence of subtractions in the uncomputation stage is shown in series, but can be parallelized across work qubits in order to minimize the depth of this stage.

4 Quantum Montgomery Multiplication

Montgomery residue arithmetic [1] is a technique for efficient multi-precision modular arithmetic, ubiquitous in applications such as hardware cryptography [28]. In the Montgomery framework, integers are mapped to a residue representation in which multiplication modulo N requires calculations under a computationally friendly auxiliary radix R. While the initial mapping of inputs to the Montgomery representation is not free, the constant overhead is quickly overcome by the efficiency of Montgomery multiplication when multiple calculations are required.

In our application, the advantage of moving to a power-of-two modulus is in flipping the reduction stage. While the Q-DIV procedure outlined in Section 3 consists of quantum additions conditioned on the most significant bits of the product register, the corresponding Montgomery operator requires only the least significant bits. This rearrangement has particularly profound advantages in the application of quantum Fourier-basis arithmetic. As in Section 3, we are able to reduce the asymptotic complexity of quantum modular multiplication to that of a single non-modular multiplication. Remarkably, it turns out that the overhead incurred in mapping to and from the Montgomery representation can be relegated entirely to classical precomputation, making our construction immediately advantageous in terms of quantum overhead.

4.1 Classical Montgomery residue arithmetic

Montgomery residue arithmetic provides an efficient computational framework for extended modular arithmetic under a fixed modulus N. Given an auxiliary radix $R = b^m$ such that $\gcd(b, N) = 1$, we define the unique N-residue representation of an integer $x \in \mathbb{Z}_N$,

$$x' \triangleq xR \bmod N. \tag{14}$$

The N-residues $\{x' \mid 0 \leq x < N\}$ form a complete residue system such that x represents the residue class containing $xR^{-1} \bmod N$. Distributivity ensures the usual addition, subtraction, and negation operations behave normally in this representation:

$$(x \pm y)' \equiv x' \pm y' \pmod{N} \tag{15}$$

However, the N-residue representation of the product xy is,

$$(xy)' \equiv (xR)(yR)R^{-1} \equiv x'y'R^{-1} \pmod{N}. \tag{16}$$

Multiplication in the N-residue representation therefore differs from the standard case, requiring the incorporation of the additional factor of $R^{-1} \pmod{N}$. The new operation is known as the Montgomery product [1],

$$\textsc{MonPro}(x', y' \mid N, R) \triangleq x'y'R^{-1} \bmod N. \tag{17}$$

We can now perform modulo-N arithmetic in the N-residue system. After initially mapping input values to their respective residue representations, we proceed with calculations as in the standard case, but with each modulo-N multiplication mapped to the corresponding Montgomery product. The result of each operation acting on N-residues is then also an N-residue, and so extended arithmetic can be performed within the residue system without further conversion steps until we finally map results back to their standard $\mathbb{Z}_N$ representations.

4.1.1 Montgomery reduction

Mirroring standard modular multiplication, the Montgomery product can be decomposed into a non-modular multiplication and a *Montgomery reduction*,

$$\text{REDC}(t \mid N, b^m) \triangleq tb^{-m} \bmod N, \tag{18}$$

where $b^m = R$ is the chosen auxiliary radix. Given $\gcd(N, b) = 1$, $\text{REDC}(t \mid N, b^m)$ provides a unique representation of $t \bmod N$. It can be efficiently computed from $t < b^m N$ due to the identity,

$$tb^{-m} \equiv (t - uN)/b^m \pmod{N}, \tag{19}$$

where,

$$u \triangleq tN^{-1} \bmod b^m, \tag{20}$$

such that, for co-prime N and b, u uniquely solves,

$$t - uN \equiv 0 \pmod{b^m}, \tag{21}$$

ensuring that $(t - uN)$ is divisible by b^m.[1]

The right hand side of Equation (19) is bound by $-N < (t - uN)/b^m < t/b^m$, such that its maximal value decreases toward zero for increasing m. Taking $m \geq \lceil \log_b(t/N) \rceil$, this limit becomes,

$$-N \leq (t - uN)/b^m < N, \tag{22}$$

enabling the computation of Montgomery residue $tR^{-1} \equiv (t-uN)/b^m \pmod{N}$ with a single comparison and corrective addition. We refer to this term as the m-digit Montgomery estimate of t,

$$\text{MonEst}(t \mid N, b^m) \triangleq (t - uN)/b^m, \tag{23}$$

and subdivide Montgomery reduction into independent *estimation* and *correction* stages (as in Algorithm 1).

Choosing $b = 2$, the division by b^m in Equation (23) becomes computationally trivial in binary architectures. Due to Equation (21), we can then compute the estimate while circumventing the explicit calculation of u. Re-expressing Equation (21) as a multiply-accumulate operation,

$$t - uN = t - \sum_{k=0}^{m-1} 2^k u_k N \equiv 0 \pmod{2^m}, \tag{24}$$

we find that each subtraction of $2^k u_k N$ is the final operation affecting the kth bit of the accumulator, and therefore must clear that bit. Beginning with the LSB ($k = 0$), we can therefore ignore u_k and simply subtract $2^k N$ if bit k of the accumulator needs to be cleared. That is, each bit u_k of u is equivalently the kth bit of the accumulator immediately prior to the corresponding conditional subtraction of $2^k u_k N$.

As each step ultimately clears a new bit of the accumulator, we can also right-shift the register by a single bit without loss of information. As described in Algorithm 1, lines 2-6, after m such iterations we have computed the Montgomery estimate $(t - uN)/2^m$, requiring only the m conditional subtractions necessary to compute $(-uN)$ and m computationally trivial right-shifts, and sidestepping the full-register comparison operations required of standard modular reduction. Combined with a single modulo-N correction (lines 7-9, in which we finally add N if the estimate is negative), we have constructed the binary algorithm for Montgomery reduction presented in Algorithm 1.

4.2 Quantum Montgomery Reduction

Given the initial construction of $|t\rangle_{n+m}$ outlined in Section 3.1, we now introduce a reversible Montgomery reduction operator in imitation of the Q-DIV operator defined in Equation (7),

$$|0\rangle |t\rangle_{n+m} \xrightarrow{\text{Q-REDC}(N,2^m)} \left|t2^{-m} \bmod N\right\rangle_n \left|tN^{-1} \bmod 2^{m+1}\right\rangle_{m+1}, \tag{25}$$

While the Montgomery reduction $(t2^{-m} \bmod N)$ is not unique for an unknown $t \geq N$, the $\text{Q-REDC}(N, 2^m)$ operation is bijective iff $t < 2^{m+1}N$ and $\gcd(N, 2) = 1$ by the Chinese remainder theorem.

[1] Note that in the above we have deviated slightly from the typical construction (for example, as presented in [1]), in which the estimate is bound by the range $[0, 2N)$. By rearranging signs slightly, we have shifted the bounds to the $(-N, N)$ range presented, which will serve to (very slightly) simplify the quantum construction we introduce below.

Algorithm 1 Classical Montgomery reduction algorithm, REDC($t \mid N, 2^m$)

```
Input: Modulus $N$, integers $t$, $m$ s.t. $t < N2^m$
Output: $S = t2^{-m} \bmod N$, $u = -tN^{-1} \bmod 2^m$
 1: $S \leftarrow t$
 2: for $k \leftarrow 0$ to $m-1$ do                # Estimation stage
 3:     $u_k \leftarrow S \bmod 2$
 4:     $S \leftarrow S - u_k \cdot N$
 5:     $S \leftarrow S/2$
 6: end for

 7: if $S < 0$ then                               # Correction stage
 8:     $S \leftarrow S + N$
 9: end if

10: return $S$
```

As in the classical procedure, we split the quantum Montgomery reduction operation into distinct *estimation* and *correction* stages. Mirroring Section 3, we will then couple the Q-REDC operator the standard initial multiplication stage (computing $|t\rangle$) and a final *uncomputation* stage (clearing $|tM^{-1} \bmod 2^{m+1}\rangle$) in order to construct a quantum Montgomery multiplication operator, Q-MonPro.

4.2.1 Estimation Stage

We first compute the Montgomery estimate, $(t - uN)/2^m$. Because this alone is not a unique representation of t, we preserve the m bits of u that naturally fall out of the classical estimation procedure, arriving at the bijective mapping,

$$|0\rangle \, |t\rangle_{n+m} \longrightarrow |(t-uN)/2^m\rangle_{n+1} \, |u\rangle_m \, . \tag{26}$$

By Equation (22), we require $n+1$ bits to represent the estimate state $|(t-uN)/2^m\rangle_{n+1}$, necessitating a single ancillary bit in addition to the $n+m$ bits holding $t < nN$.

We proceed as in Algorithm 1. Prior to the kth iteration of the estimation stage, the LSB of the accumulation register is equivalently the kth bit of u, or u_k (Algorithm 1, line 3). Classically, u_k is then used to condition the subtraction of N, so as to clear the LSB of the accumulator (line 4). This represents a two-to-one operation, necessitating the creation of a garbage bit in the reversible case. However, after each iteration, we know also that the newly cleared LSB will be immediately shifted out (line 5); we can therefore consider only the effect of subtraction on the remaining bits of the register. The subtraction occurs only when the accumulator is odd (and N is odd by design), so no borrow will be generated by the first bit of the subtraction. It is then equivalent to subtract $\lfloor N/2 \rfloor = (N-1)/2$ from the truncated register, conditioned on the LSB $|s_0\rangle = |u_k\rangle$:

$$|S\rangle_w = |s_{w-1}...s_1\rangle_{w-1} \, |s_0\rangle = |\lfloor S/2 \rfloor\rangle_{w-1} \, |u_k\rangle \longrightarrow |\lfloor S/2 \rfloor - u_k \cdot \lfloor N/2 \rfloor\rangle_{w-1} \, |u_k\rangle \, . \tag{27}$$

In this way, we avoid accumulating new ancilla bits with each reduction step. Iterating through $k = 0, ..., m-1$, we compute the Montgomery estimate $(t - uN)/2^m$ with n controlled subtractions. The garbage information created in this sequence is simply the m bits of u, which are computed in the place of the m least significant bits of the input state $|t\rangle_{n+m}$. As shown in Figure 7, the sequence of in-place subtractions mirrors that of the division case (Section 3, Figure 5), but with adders controlled by the least significant bits of the product register. Both reduction procedures have the same depth (in quantum adders)–while Montgomery reduction sidesteps the trial subtractions required in the Q-DIV operation, the implementation of controlled, in-place quantum adders requires a pair of out-of-place adders identical to the paired adders of the division operation.

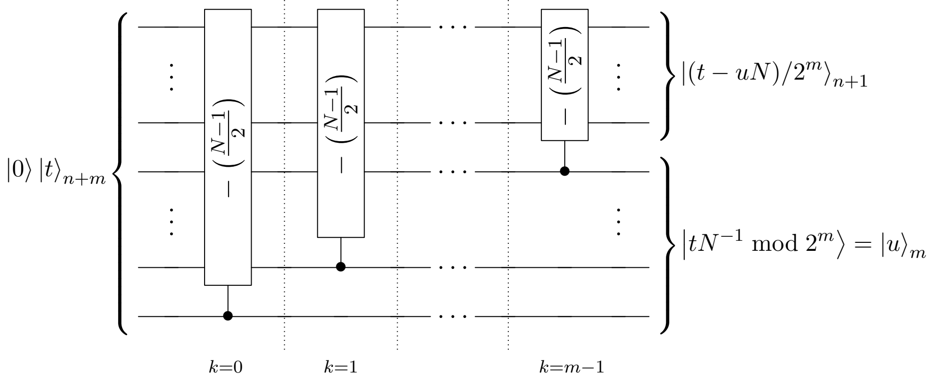

$|0\rangle\,|t\rangle_{n+m}$
$-\left(\frac{N-1}{2}\right)$
$|(t-uN)/2^m\rangle_{n+1}$
$\left|tN^{-1} \bmod 2^m\right\rangle = |u\rangle_m$
$k{=}0$
$k{=}1$
$k{=}m{-}1$

Figure 7: Estimation stage of the quantum Montgomery reduction algorithm (Section 4.2.1), Q-MonEst$(N, 2^m)$. Note the parallel to the division-based procedure (Figure 5); where the latter computes the quotient q with trial subtractions and re-additions conditioned on the MSBs of the accumulation register, Montgomery reduction allows for the computation of u from the LSBs of the register.

4.2.2 Correction stage

The second piece of the classical Montgomery reduction algorithm is a single modulo-N correction (Algorithm 1, lines 7-9). For $-N < (t - uN)/2^m < N$, this correction requires a single controlled addition of N conditioned on the sign of the estimate. Labeling the sign bit $|s_\pm\rangle$, we perform,

$$|(t-uN)/2^m\rangle_{n+1} \longrightarrow |s_\pm\rangle\,|(t-uN)/2^m + s_\pm \cdot N\rangle_n\,, \tag{28}$$

where by Equation (19) the final term is equivalently $|t2^{-m} \bmod N\rangle_n$.

We are now left with the sign bit $|s_\pm\rangle$ as garbage. For odd N, the conditional addition of N must change the LSB of the register. Defining the LSBs before and after the modular reduction,

$$p_\oplus \triangleq \big(t2^{-m} \bmod N\big) \bmod 2, \tag{29}$$

$$s_\oplus \triangleq (t-uN)/2^m \bmod 2, \tag{30}$$

we therefore find $s_\pm \oplus p_\oplus = s_\oplus$. By negating the sign bit $|s_\pm\rangle$, conditional on $|p_\oplus\rangle$, with a single CNOT gate (as in Figure 8), we return it to the pre-addition LSB $|s_\oplus\rangle$.

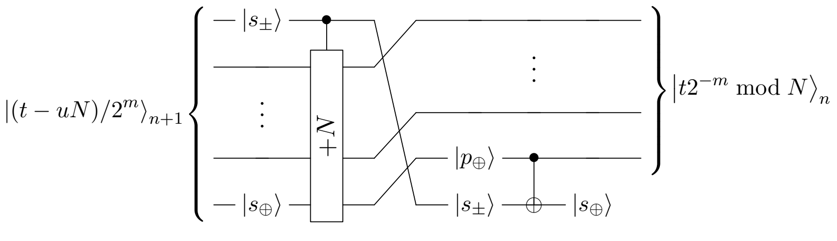


Figure 8: Quantum circuit demonstrating the correction stage of a quantum Montgomery reduction algorithm (Section 4.2.2), assuming $-N < (t-uN)/2^m < N$.

We can then re-express,

$$s_\oplus \equiv (t-uN)/2^m \equiv (\tilde{u}-u)/2^m \pmod 2. \tag{31}$$

where,

$$\tilde{u} \triangleq tN^{-1} \bmod 2^{m+1}, \tag{32}$$

descends from an $(m+1)$-bit Montgomery estimate $(t-\tilde{u}N)/2^{m+1}$. In this form, $|s_\oplus\rangle$ can be concatenated with $|u\rangle_m$ and equivalently described,

$$|s_\oplus\rangle\,|u\rangle_m = |2^m s_\oplus + u\rangle_{m+1} = |\tilde{u}\rangle_{m+1} = \left|tN^{-1} \bmod 2^m\right\rangle_{m+1}, \tag{33}$$

completing the Q-REDC$(N, 2^m)$ operation introduced in Equation (25).

4.3 Uncomputation

In order to construct a quantum Montgomery multiplier from the Q-REDC$(N, 2^m)$ operator, we must finally uncompute the auxiliary output state $|\tilde{u}\rangle_{m+1}$, mirroring the uncomputation of the quotient in Section 3. Given the partial products composing t, we can express $\tilde{u}$,

$$tN^{-1} \equiv \sum_{k=0}^{n-1} y_k \big(2^k X \bmod N\big) N^{-1} \pmod{2^{m+1}}. \tag{34}$$

We can therefore classically precompute the classical addends $((2^k X \bmod N)N^{-1} \bmod 2^{m+1})$ for $k = 0, ..., n-1$, and use n controlled $(m+1)$-bit subtractions, conditioned on the bits of $|y\rangle_n$, to clear the register. Identically to Section 3.3, the narrow register enables parallelization of the quantum adders to an overall depth of $\mathcal{O}(\log_2^2 n)$.

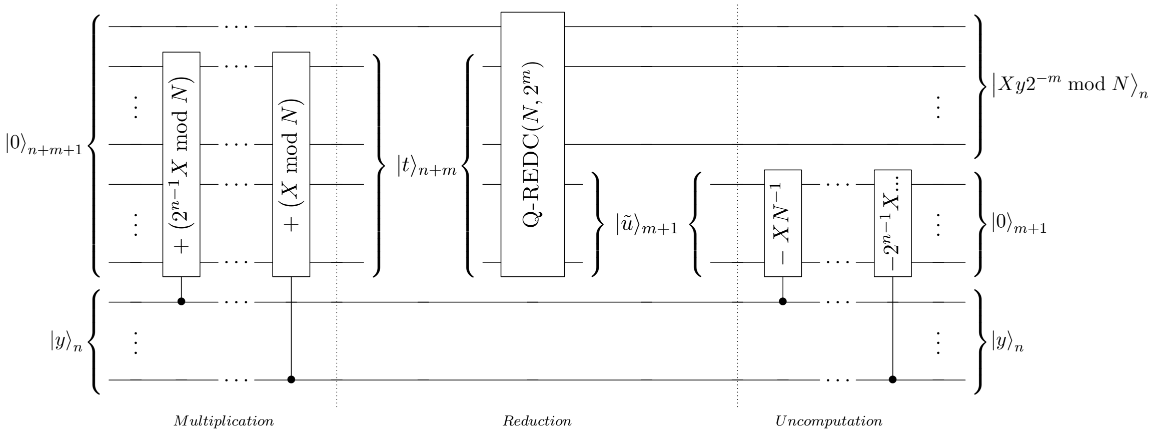


Figure 9: Out-of-place quantum Montgomery multiplier Q-MonPro$(t \mid N, 2^m)$, comprising the initial computation of $|t\rangle_{n+m}$, estimation (Section 4.2.1) and correction (Section 4.2.2) stages of Q-REDC$(N, 2^m)$, and a final uncomputation of $|\tilde{u}\rangle_n$.

4.4 Modular Multiplication via Quantum Montgomery Reduction

In the case of quantum-classical multiplication, we can use the quantum Montgomery multiplier developed here to compute a product where the quantum register remains in the standard representation. Observing the Montgomery product of an N-residue X' and a value y in standard representation,

$$\textsc{MonPro}(X', y \mid N, R) = (XR \bmod N)yR^{-1} \bmod N = Xy \bmod N, \tag{35}$$

we find the modular product $Xy \bmod N$ returned in standard representation. Classically, this would represent a disadvantage for most operations: we now need to remap the result to an N-residue representation for subsequent calculations. In the quantum-classical case, however, we can precompute the residue of the classical multiplicand off-line, and use it to compute the residue,

$$t' \triangleq \sum_{k=0}^{n-1} y_k (2^k X' \bmod N), \tag{36}$$

from $|y\rangle$ identically to the usual product $|t\rangle_{n+m}$. Applying Q-REDC$(N, 2^m)$ to $|t'\rangle_{n+m}$, we compute the modular product $|Xy \bmod N\rangle_n$ in standard representation while relegating conversion overhead of Montgomery multiplication to classical precalculation.

4.4.1 Montgomery Multiplication with Two Quantum Inputs

We can also define a quantum Montgomery multiplier taking as input two n-bit quantum values. This requires adapting the initial multiply so that it accepts two quantum inputs, from which we can compute $|t\rangle_{n+m}$ with n quantum shift-and-reduce operations, as defined in Section 2. The Q-REDC$(N, 2^m)$ algorithm described in Section 4.2 acts only on $|t\rangle_{n+m}$ and is independent of the individual multiplicands, and therefore goes unchanged. However, the final uncomputation of $|\tilde{u}\rangle_{m+1}$ requires some modification: we can no longer precompute the addends $\{(2^k x \bmod N)N^{-1} \bmod 2^{m+1}, k = 0, ..., n\}$. Instead, noting Equation (21), we perform an in-place multiplication of $|\tilde{u}\rangle_{m+1}$ by the classical value N (requiring $m = \lceil \log_2 n \rceil$ adders). We are then left with the truncated product $|t\rangle_{m+1}$, which we can clear by reversing the n adders of the initial multiplication (but truncated to $m+1$ bits). The reversed sequence will further undo the shift of $|x\rangle_n$ in the initial multiply:

$$\begin{aligned}
|0\rangle_{n+m+1} |x\rangle_n |y\rangle_n &\xrightarrow{\text{Q-MAC}} |0\rangle_1 |t\rangle_{n+m} \qquad\qquad |2^n x \bmod N\rangle_n |y\rangle_n \\
&\xrightarrow{\text{Q-REDC}(N,2^m)} \left|xy2^{-m} \bmod N\right\rangle_n |\tilde{u}\rangle_{m+1} |2^n x \bmod N\rangle_n |y\rangle_n \\
&\xrightarrow{\text{Q-MUL}(N)} \left|xy2^{-m} \bmod N\right\rangle_n |t\rangle_{m+1} |2^n x \bmod N\rangle_n |y\rangle_n \\
&\xrightarrow{\text{Q-MAC}^\dagger} \left|xy2^{-m} \bmod N\right\rangle_n |0\rangle_{m+1} |2^n x \bmod N\rangle_n |y\rangle_n .
\end{aligned} \tag{37}$$

5 Quantum Barrett Multiplication

For classical modular arithmetic the Barrett reduction [2] is beneficial because, for repeated reductions using the same modulus, it reduces the number of divisions by that modulus and replaces them with simpler multiply and shift operations. The standard method to reduce the number t modulo N would be to calculate $q = \lfloor t/N \rfloor$ and then calculate the reduced value as: $t - qN$. The main idea behind the Barrett reduction is to calculate a fixed-point fractional factor representing the division by the modulus and then use this factor many times. The only divisions involving t can be picked to be constant powers of the machine word size and therefore can be implemented with shifts. Because of limited precision of the fixed-point factor, the Barrett reduction may not reduce t completely to a value $< N$. However, we can set the precision of the factor so that at most one additional subtraction by N is required. Because of this we can view the Barrett reduction as an approximate reduction technique.

With our quantum circuit we implement all operations on qubits using binary fixed-point logic. Therefore, shift operations by a power of 2 just redefine the fixed-point in our calculation. Additionally, the fixed-point fractional factor is a fixed classical value and can be pre-calculated offline. The quantum circuit to calculate q is reduced to a quantum multiplication by a classical value. As is the case for the classical Barrett reduction, the quantum circuit calculates an approximate reduction factor, and we will want to carefully set the width of each of the individual operations in the reduction to bound the error and reduce the total number of gates required. Completing the reduction is one of the challenges in constructing a reversible circuit for it, and is one of the main differences between our implementation and that from [4]. We show how to perform a complete reduction, whereas in [4] they allow the case of a partial reduction and argue that doing so has an insignificant impact on the fidelity of the circuit.

5.1 The Barrett Reduction

The Barrett reduction [2] of an arbitrary number t is defined as,

$$\mathrm{REDC}(t \mid N) = t - \tilde{q}N \bmod N, \tag{38}$$

where

$$\tilde{q} = \left\lfloor \left\lfloor \frac{t}{b^{k-1}} \right\rfloor \frac{\mu}{b^{k+1}} \right\rfloor \quad \mu = \left\lfloor \frac{b^{2k}}{N} \right\rfloor \tag{39}$$

and the mod$N$ involves at most one subtraction of N because the parameters b and k can be picked to ensure that $REDC(t) < 2N$. Typically b is picked to be a small power of 2 and k must be picked so that $b^{k-1} \leq N < b^k$. The only operation involving the value t is a multiplication. If b is picked to be a power of two then the factors of b^{k-1} and b^{k+1} appearing in the denominators can be implemented as binary shifts, via redefinition of the fixed-point binary number.

For our multiplier the value to be reduced is $t = Xy$. Further, we pick k based on the value N, and we pick $b = 2$, therefore k is just the bit-width of N, which we will denote as n. The value Xy is calculated as the sum over i as: $\sum y_i(2^i X \bmod N)$. Because the shifted value is reduced before adding to the running sum, the total number of bits required for the product is $n + \log_2(n)$. In the calculation of $\tilde{q}$ we only use the most significant bits of Xy and μ. To understand the impact of truncation it it useful to look at the full-precision quotient q, defined as:

$$q = \left\lfloor \frac{Xy}{2^{n-1}} \nu \right\rfloor \quad \nu = \frac{1}{2^{n+1}} \left(\frac{2^{2n}}{N} \right), \tag{40}$$

where ν corresponds to the shifted μ and because $2^{n-1} < N < 2^n$ we have $1/2 < \nu < 1$. We can now write q to separate the calculated values from the truncated ones,

$$q = \left\lfloor \frac{\widetilde{Xy} + (Xy)_t}{2^{n-1}} (\tilde{\nu} + \nu_t) \right\rfloor, \tag{41}$$

where $\tilde{a}$ denotes the retained approximate value, and a_t is the truncated portion of a value. Separating the computed terms from the truncated ones we have:

$$q = \left\lfloor \frac{\widetilde{Xy}}{2^{n-1}}\tilde{\nu} + \frac{(Xy)_t\nu}{2^{n-1}} + \frac{\widetilde{Xy}\nu_t}{2^{n-1}} \right\rfloor = \left\lfloor \tilde{q} + \frac{(Xy)_t\nu}{2^{n-1}} + \frac{\widetilde{Xy}\nu_t}{2^{n-1}} \right\rfloor. \tag{42}$$

If we bound the truncated terms to be less than 2 then only a single extra adjustment will be required. We could use the upper bits of Xy to calculate $\tilde{q}$, however, it will be useful for our quantum implementation to have an independent $\widetilde{Xy}$ therefore we would like to minimize the width of this calculation. If we use the n_k upper bits of each term in the sum then we can bound the first truncated term as:

$$\frac{(Xy)_t\nu}{2^{n-1}} < n\frac{2^{n_t}1}{2^{n-1}} = \frac{2^{\log_2(n_t)}2^{n_t}}{2^{n-1}}. \tag{43}$$

Where $n_t = n - n_k$ is the number of bits truncated from each term, and we have taken the upper bound of $\nu = 1$. If we pick n_t such that: $\log_2(n) + n_t < n - 1$ then the error term will be < 1. This implies that each term must be $n_k = \log_2(n) + 1$ bits, and the total approximate sum of n values will require $2\log_2(n) + 1$ bits.

For the second truncated term, if we use n_v bits for ν we can bound this term as,

$$\frac{\widetilde{Xy}\nu_t}{2^{n-1}} < \frac{n2^n2^{-n_v}}{2^{n-1}} < \frac{2^{\log_2(n)+n-n_v}}{2^{n-1}}. \tag{44}$$

We then need to pick $n_v > \log_2(n)+1$ to ensure that this term is less than 1. The resulting bit widths for $\widetilde{Xy}$ and ν result in an $2\log_2(n)+2$ by $\log_2(n)+1$ bit multiplication to calculate $\tilde{q}$. We can truncate the $\widetilde{Xy}$ values used to calculate $\tilde{q}$ to $\log_2(n)+1$ bits and therefore we need a register of length $2\log_2(n)+2$ to hold $\tilde{q}$.

5.2 Quantum Modular Multiplier using the Barrett Reduction

In Algorithm 2 we describe the algorithm to compute the modular product of two numbers using the Barrett reduction described in the previous section. A quantum circuit to calculate the out-of-place modular product of either a quantum register with a classical constant or two quantum registers can be constructed directly from this algorithm. In the following discussion we will just describe the case of a quantum product with a classical constant. We will return to the implications of a full quantum multiply in Section 5.2.1.

The Barrett multiplier uses one input register, one output register, two work registers, and one single qubit flag. The out-of-place multiplier performs the following operation:

$$|y\rangle_{\,n}\,|0\rangle_{\,n+m}\,|0\rangle_{\,2m}\,|0\rangle_{\,2m}\,|0\rangle_{\,1} \longrightarrow |y\rangle\,|yX \bmod N\rangle\,|0\rangle\,|0\rangle\,|0\rangle. \quad (m = \log_2(n)+1) \tag{45}$$

In Steps 1 and 2 of the algorithm we calculate the full and approximate products and produce the state

$$|y\rangle\,|Xy\rangle\,|\widetilde{Xy}\rangle\,|0\rangle\,|0\rangle. \tag{46}$$

The full product (Xy) requires $\mathcal{O}(n^2)$ basic gates and constitutes the majority of the operations in the entire circuit. Calculating an approximate product eliminates the need to re-compute the full product later when we need to clear the reduction factor $\tilde{q}$. In Steps 3 and 4 of the algorithm we calculate the approximate reduction factor and use it to reduce the full product in-place producing the state:

$$|y\rangle\,|Xy - \tilde{q}N\rangle\,|\widetilde{Xy}\rangle\,|\tilde{q}\rangle\,|0\rangle. \tag{47}$$

As discussed above, the state reduced using $\tilde{q}$ may be greater than N and therefore one more reduction by N may be required. We perform this reduction, however, doing so results in a one-bit flag indicating whether the additional reduction was required. At this point we have the following state:

$$|y\rangle\,|Xy \bmod N\rangle\,|\widetilde{Xy}\rangle\,|\tilde{q}\rangle\,|adj\rangle. \tag{48}$$

and we have the reduced product, however, we have two registers with garbage and the one-bit adjustment flag that need to be cleared. The two registers, containing $\widetilde{Xy}$ and $\tilde{q}$, can be cleared simply by reversing steps 2 and 3 of the algorithm, however the adjustment bit must be cleared in some other way. If we add back $\tilde{q}N$ to P then this register contains $|Xy - adjN\rangle$. If we subtract $\widetilde{Xy}$ from this register we obtain $|(Xy)_t - adjN\rangle$. But in Equation (43) we bounded the truncation term $(Xy)_t < 2^{n-1} < N$ and therefore if $(Xy)_t - adjN < 0$ this indicates that an adjustment has occurred. This fact can be used to clear the adjustment bit. We also note that we only need to compute the high-order bits for the addition done in step 9 of the algorithm.

```
Algorithm 2 BarPro($X, y \mid N$)
Input: Modulus $N$, integers $X, y < N$
Output: Integer $P = yX \bmod N$
 1: $S \leftarrow Xy$                                  # calculate the full product
 2: $\widetilde{Xy} \leftarrow app(Xy)$                # calculate an approximate product
 3: $\tilde{q} \leftarrow \widetilde{Xy}\tilde{\nu}$   # calculate the approximate reduction factor
 4: $S \leftarrow S - \tilde{q}N$                      # reduce S s.t. $S < 2N$
 5: if $S \geq N$ then
 6:     $S \leftarrow S - N$                           # reduce by N if required
 7:     $adj \leftarrow 1$                             # reduction produces one bit of garbage
 8: end if
 9: $S \leftarrow S + \tilde{q}N$                      # $S = Xy - adjN$
10: if $S_{[n+\log_2(n):n-\log_2(n)]} - \widetilde{Xy} < 0$ then
11:     $adj \leftarrow adj \oplus 1$                  # clear adjustment flag
12: end if
13: $S \leftarrow S - \tilde{q}N$                      # reset to modular product
14: $\tilde{q} \leftarrow 0$                           # reverse 3 to clear $\tilde{q}$
15: $\widetilde{Xy} \leftarrow 0$                      # reverse 2 to clear $\widetilde{Xy}$
```

From Equation (45) we see that the out-of-place modular multiplication of a quantum register by a constant requires $2n + 5m + 1 = 2n + 5\log_2(n) + 6$ total qubits. This is compared to the $2n$ qubits required by the standard method that utilizes modular adders. The additional overhead of $5\log_2(n) + 6$ is small for realistic sized multipliers.

In Table (3) we show the overhead in terms of the size and total number of addition operations required per step in Algorithm 2. For comparison the standard modular addition based adder would require $3n$ adders each of width n. The total number of gates is linear in the number and width of the adders, therefore, the product of the adder-width times the number of adders gives to first-order the number of gates required. For the Barrett multiplier this product is: $n^2 + 14n\log_2(n) + n + 17(\log_2(n))^2 + \log_2(n)$, compared to the standard multiplier with $3nn = 3n^2$. For realistic sized multipliers the n^2 term dominates for both adders and the Barrett multiplier provides close to a factor of 3 fewer gates than the standard method. The circuit depth of the multipliers will depend on the implementation of the adders used as well as how the individual steps of the multiplier overlap.

5.2.1 Quantum Barrett multiplication with two quantum inputs

The Barrett reduction method can be extended to a full-quantum multiplier, i.e., one where both inputs are contained in quantum registers. For this adder we can either add the shifted terms $2^i x$ directly or reduce them modulo N before adding them. Since x is a quantum value, reducing them would require the shift and reduce circuit described in Section 2.5, but the operation of the Barrett reduction is the same as the case when one input is classical. If we assume that the accumulated shifts must be reversed at the end of the multiplier then $2n$ shifts are required per out-of-place multiplication. Therefore the full-quantum Barrett Multiplier requires $3n$ additions compared to the $3n + 2n = 5n$ additions that would be required by the corresponding standard quantum-quantum modular multiplier. Adding the shifted

	Width of Adder	
Step	$n + \log_2(n)$	$\log_2(n)$
1	n	
2		$2n$
3		$4\log_2(n)$
4	$3\log_2(n)$	
6	1	
9	$3\log_2(n)$	
13	$3\log_2(n)$	
14		$4\log_2(n)$
15		$2n$
	$n + 9\log_2(n) + 1$	$4n + 8\log_2(n)$

Table 3: Full-width and log-width adders required by the steps of Algorithm 2 The n adders of width $n + \log_2(n)$ required to calculate the full product dominate the required resources

terms directly would eliminate the $2n$ reduction steps, but would require a $2n - bit$ product register and would require higher precision in the Barrett reduction. As in the case of the standard modular multiplier, constructing an in-place full-quantum multiplier would require calculating a multiplicative inverse, which would dominate the total cost of the multiplier.

6 Modular multiplication with Quantum Fourier Arithmetic

Because of the elimination of division from the modular reduction step our Barrett and Montgomery modular multipliers constructions are uniquely amenable to arithmetic in the quantum Fourier number basis. In the Fourier basis, number states are represented by the quantum Fourier transform (QFT) of their binary state,

$$|y\rangle_n^{\Phi} \triangleq \mathrm{QFT}\,|y\rangle_n = \bigotimes_{k=0}^{n-1} \left\{ \cos\!\left(\frac{y\pi}{2^k}\right)|0\rangle - i\sin\!\left(\frac{y\pi}{2^k}\right)|1\rangle \right\}, \tag{49}$$

where we have commuted and dropped Hadamard gates from the QFT definition (see the Appendix for details). Arithmetic in this representation circumvents the ancillary work qubits and data dependencies required by the carry bits needed for binary-basis arithmetic, absorbing these bits into the continuous state of each qubit. Fourier-basis addition is decomposed into independent, commutable rotations acting on each qubit in the register independently, enabling large-scale parallelization of arithmetical operations with minimal ancilla and on a variety of computational topologies (e.g. a nearest-neighbor architecture) [29, 30, 18].

The bottleneck of quantum Fourier-basis arithmetic is in the implementation of quantum control. The continuous state $|y\pi/2^k\rangle^{\Phi}$ of the kth qubit of a Fourier-basis register $|y\rangle_n^{\Phi}$ contains information about the bit y_k as well each bit $y_{j<k}$ less significant than y_k. In order to condition a quantum operation on y_k, we therefore require a k-bit $\mathrm{QFT}^{\dagger}$ in order to extract the single bit as a Z-eigenstate. Quantum Fourier arithmetic is generally characterized by these repeated transformations to and from the Fourier basis.

This limitation introduces a significant computation discrepancy between quantum Fourier-basis multiplication and division. Each reduction step composing a typical division operation comprises a trial subtraction followed by a controlled re-addition conditioned on the sign bit (MSB) of the difference. For a quantum register in Fourier representation, each such comparison requires a full-register $\mathrm{QFT}^{\dagger}$ in order to extract the sign as a binary state, and subsequent QFT to return the rest of the register to the Fourier basis prior to the re-addition. In addition to the overhead of the QFTs themselves, each transform acts as a computational barrier point, inhibiting the parallelization of sequential adders. A single modular adder constructed as in Section 2.4 then requires two $\mathrm{QFT}^{\dagger}$-QFT pairs (for the two embedded comparisons), totaling $4n$ full-register QFT-like operations for an out-of-place quantum modular multiplier constructed from Fourier modular adders [7, 8].

6.1 Quantum Fourier Division

The utility of Fourier-basis arithmetic for quantum modular multiplication can be improved by separating the component multiplication and division stages. The initial multiplication stage, comprising only controlled additions to an accumulation register, can be implemented with parallelized Fourier adders, such that from a binary input state $|y\rangle_n$ we accumulate the Fourier-basis state $|t\rangle_{n+m}^{\Phi}$,

$$|0\rangle_{n+m}\,|y\rangle_n \;\longrightarrow\; |t\rangle_{n+m}^{\Phi}\,|y\rangle_n\,, \tag{50}$$

with a total parallelized depth of $(n+m)$ controlled rotation gates.

The quantum division operator (Q-DIV) defined in Section 3 consists of $m = \lceil \log_2 n \rceil$ reduction steps. Using Fourier-basis arithmetic, each trial subtraction must be followed by n-bit $\mathrm{QFT}^{\dagger}$ and QFT operations in order to extract the resulting sign bit. After m steps, the remainder ($t \bmod N$) is held in Fourier representation, while the quotient, computed by inverting the m extracted sign qubits, is constructed in binary:

$$|t\rangle_{n+m}^{\Phi} \xrightarrow{\Phi\text{-DIV}(N)} |q\rangle_m\,|t \bmod N\rangle_n^{\Phi}\,. \tag{51}$$

In order to construct the in-place modular multiplier described in Section 2.2, both outputs of the out-of-place operator must share the same representation. We therefore apply one more n-bit $\mathrm{QFT}^{\dagger}$ to the remainder $|t \bmod N\rangle_n^{\Phi}$, leaving both it and the input $|y\rangle_n$ in binary representation.

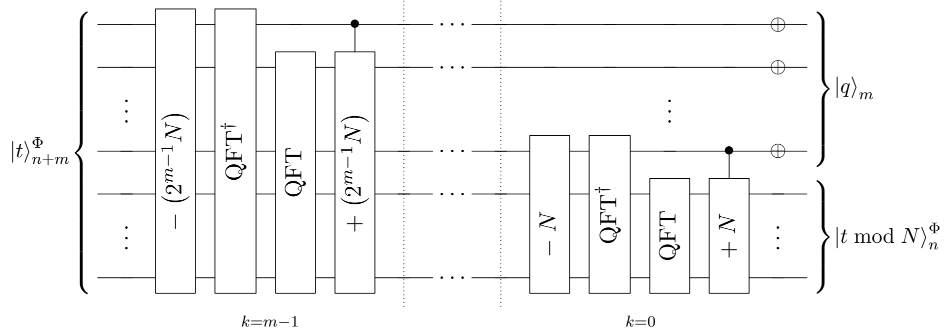


Figure 10: Φ-DIV(N) circuit, incorporating the intermediate QFT† and QFT operators required to extract the sign after each trial subtraction. The quotient, constructed by inverting the extracted sign bits, is computed in binary representation, while the remainder is output in the Fourier basis.

We finally uncompute the quotient register as in Section 3.3:

$$|q\rangle_m \xrightarrow{\Phi\text{-MUL}(N)} |qN\rangle^{\Phi}_m \longrightarrow |qN + (t \bmod N)\rangle^{\Phi} = |t\rangle^{\Phi}_m \longrightarrow |0\rangle_m . \tag{52}$$

Using the Φ-MUL(N) operator defined in Appendix A.5, we first multiply the register by N (modulo-2^m) in-place, while simultaneously transforming the result to its Fourier representation. We then can add the remainder and uncompute the resulting $|t\rangle^{\Phi}_m$ register with a combined $(n+m)$ width-m Fourier adders, controlled by the m LSBs of the output register and all n bits of $|y\rangle_n$. The gates composing the Fourier uncomputation stage can be overlapped with gates in either the Φ-DIV(N) operation or the final QFT†.

6.1.1 Analysis

The circuit dimensions of the out-of-place modular multiplier constructed from the Q-DIV operation are broken down in Table (4). The total two-qubit gate count of the multiplier is,

$$\#(\text{gates}) = n^2 m + 3n^2/2 + 3nm - n/2 + m^2/2 - m/2, \tag{53}$$

parallelized to a depth of,

$$4nm + 2n + \mathcal{O}(m). \tag{54}$$

Though a significant speedup over Fourier modular multiplication via modular addition [7, 8], the overall depth scales as $\mathcal{O}(n \log n)$, offering no speedup over the binary-basis multiplier constructed with logarithmic-depth adders. However, we require no work qubits for the Fourier adders, so that the total operator requires only the $(2n+m)$ qubits required to hold $|t\rangle^{\Phi}_{n+m}$ alongside the input state $|y\rangle_n$. The in-place-multiplier doubles the gate count and depth requirements, and adding a quantum control add $\mathcal{O}(n)$ gates.

Table 4: Total two-qubit gates and parallelized depth of an n-bit out-of-place Fourier modular multiplier constructed from the Φ-DIV(N) operator (where $m = \lceil \log_2 n \rceil$).

Stage		Adds	QFTs	Width	Gates	Depth
Multiplication		n	0	$n+m$	n^2+nm	$n+m$
Division (Φ-DIV)		m	$2m$	n	n^2m+nm	$4nm$
Output QFT†		0	1	n	$(n^2-n)/2$	$2n$
Uncompute:	$|q\rangle_m \longrightarrow |qN\rangle^{\Phi}_m$	$m-1$	0	$1,...,m-1$	$(m^2-m)/2$	$2m$ ‡
	$\longrightarrow |t\rangle^{\Phi}_m$	m	0	m	m^2	m ‡
	$\longrightarrow |0\rangle_m$	n	0	m	nm	n ‡

‡Executed in parallel with Φ-DIV and output QFT†

The gate count of the Φ-DIV-multiplier can be further reduced if we bound the precision of the controlled rotations comprising each QFT. By eliminating rotations by angles below a determined threshold, the number of gates required for an n-bit QFT is decreased from $n^2/2$ to $\mathcal{O}(n \log n)$, while overall fidelity, of a noisy circuit, is likely improved [23]. Applying this optimization to the QFTs embedded in the Φ-DIV operation, the total gate count of the multiplier becomes,

$$\#(\text{gates, bound precision}) = n^2 + \mathcal{O}(n \log^2 n), \tag{55}$$

where the magnitude of the second-order term is determined by the desired precision and fidelity of physical operations. In this case, the asymptotic gate count of the modular multiplier is dominated by the n^2 gates of the initial multiplication. Unfortunately, this optimization does not change the depth of the QFT or the overall modular multiplier.

6.2 Quantum Fourier Montgomery Multiplication

Beginning with the Fourier state $|t\rangle^{\Phi}_{n+m+1}$ (where we extend Fourier-basis multiply described in Section 6.1 to incorporate the single ancilla necessary to hold the MSB of the quantum Montgomery estimate), we can reconstruct the quantum Montgomery reduction operator using Fourier-basis arithmetic. The bottleneck of the Φ-DIV circuit was in the extraction of the sign bit after each trial subtraction, necessitating n-bit QFT† and QFT operations due to the dependency of the MSB on the less significant bits of the Fourier register. The Montgomery reduction algorithm sidesteps the requirement of trial subtraction, instead requiring additions controlled by the LSBs of the register.

6.2.1 Estimation stage

As in the binary case, the Fourier Montgomery estimation stage is constructed from controlled Fourier subtractions and implicit right-shifts. For integral t, the LSB $|t_0\rangle^{\Phi}$ of the Fourier state $|t\rangle^{\Phi}_{n+m+1}$ ($k = 0$ term in Equation (49)) is equivalent (up to phase) to that of its binary representation. We use this bit to condition a subtraction of N from the register, ignoring the irreversible component of the subtraction affecting the control qubit. In the continuous Fourier representation, this is equivalent to subtracting the half-integer $(N/2)$ from the truncated register,

$$|t\rangle^{\Phi}_{n+m+1} = |t/2\rangle^{\Phi}_{n+m} |t_0\rangle \longrightarrow |(t/2) - t_0 \cdot (N/2)\rangle^{\Phi}_{n+m} |t_0\rangle, \tag{56}$$

where $|t_0\rangle$ is then equivalently the LSB u_0 of $|u\rangle_m$.

By design, the subtraction occurs only when t is odd ($t_0 = 1$), so that the difference is always an integer. The LSB of the truncated Fourier-basis state can therefore be used to condition the next subtraction of $(N/2)$ from the remaining bits of the register. As shown in the first stage of Figure 11, after m such iterations we are left with the Montgomery estimate in Fourier representation, while simultaneously extracting the m bits of $|u\rangle_m$ in binary:

$$|t\rangle^{\Phi}_{n+m+1} \xrightarrow{\Phi\text{-}\textsc{MonEst}(N,2^m)} |(t - uN)/2^m\rangle^{\Phi}_{n+1} |u\rangle_m. \tag{57}$$

Remarkably, we have thus far required no extra transformations.

6.2.2 Correction stage

Unfortunately, the quantum Montgomery reduction procedure does require a single modular reduction. In the correction stage, we add N to the estimate if it is negative, requiring a single $(n + 1)$-bit QFT† to extract the sign bit of the register, followed by an n-bit QFT to return the remaining bits of to their Fourier representation. As demonstrated in Figure 11, the binary sign bit is then used to control the addition of N, after which it is conditionally flipped by the LSB of the result and concatenated with $|u\rangle_m$ to form $|\tilde{u}\rangle_{m+1}$ identically to the binary case. Combined with the estimation stage, we have constructed the Fourier Montgomery reduction operator,

$$|t\rangle^{\Phi}_{n+m+1} \xrightarrow{\Phi\text{-REDC}(N,2^m)} \left|t2^{-m} \bmod N\right\rangle^{\Phi}_{n} \left|tN^{-1} \bmod 2^{m+1}\right\rangle_{m+1}, \tag{58}$$

where the reduction is returned in Fourier representation, and the garbage state $|\tilde{u}\rangle_{m+1}$ in binary.

We can then concatenate the QFT† operation with the Fourier adders comprising the preceding estimation stage. The resulting sequence of controlled rotations is identical in structure to that of a single QFT† over all $n+m+1$ qubits, and can likewise be parallelized to a depth of $2(n+m)-1$. Similarly, the controlled addition of N can be concatenated with the preceding QFT and parallelized as an $(n+1)$-qubit QFT to depth $2n-1$ controlled rotation gates.

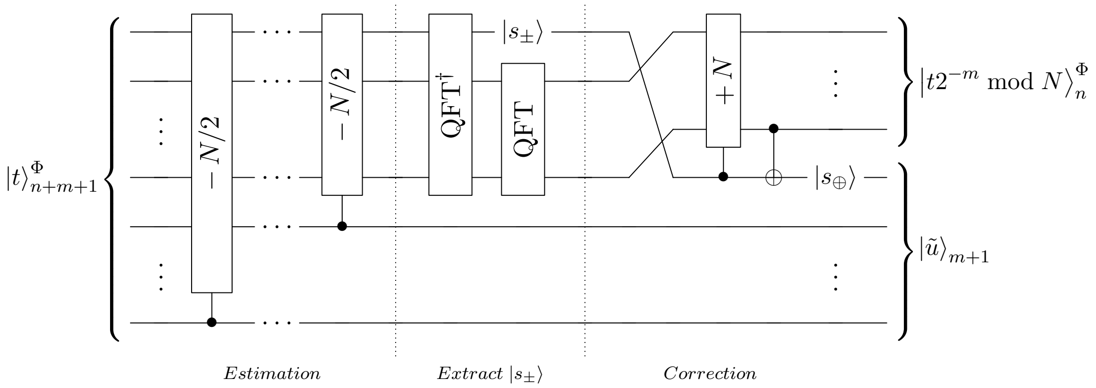


Figure 11: Φ-REDC$(N, 2^m)$ circuit, with the requisite QFT† and QFT operators in order to extract the sign bit $|s_\pm\rangle$. The estimation stage $(-N/2)$ adders and QFT† can then be sequenced like a single QFT† over $n+m+1$ qubits.

6.2.3 Uncomputation Stage

As in the case of the Φ-DIV(N) procedure, in order to construct an in-place quantum Montgomery multiplier from the Φ-REDC$(N, 2^m)$ operator, we require a final n-bit QFT† on the output so that it is returned in binary along with the input state $|y\rangle_n$. Simultaneously, we uncompute the $|\tilde{u}\rangle_{m+1}$ register. After transforming the register to its Fourier representation with an $(m+1)$-bit QFT, we can replicate the sequence of subtractions in the binary procedure (Section 4.3) with n controlled Fourier adders, each truncated to $(m+1)$ bits. Being independent of the output state, the n-gate depth of this uncomputation is dominated by the $(2n-3)$-gate depth of the concurrent QFT†, fixing the total depth of this stage to the latter.

6.2.4 Analysis

Circuit characteristics of the out-of-place Fourier Montgomery multiplier are broken down by stage in Table (5). In total, we require,

$$\#(\text{gates}) = 5n^2/2 + 3nm - n/2 + m^2/2 - m/2, \tag{59}$$

parallelized to a depth of,

$$7n + \mathcal{O}(m). \tag{60}$$

Comparing the Montgomery circuit to the division-based circuit we see than the depth of the division circuit is higher by a factor of m. This is a result of the extra QFTs required in the reduction portion of the division circuit. As with the Φ-DIV operator, if we bound the precision of the rotation gates composing each QFT, the total gate count is reduced to $n^2 + \mathcal{O}(nm)$, asymptotically equivalent to that of just the initial multiplication stage.

Table 5: Total two-qubit gates and parallelized depth of an n-bit out-of-place Fourier Montgomery multiplier (where $m = \lceil \log_2 n \rceil$).

Stage		**Adds**	**QFTs**	**Width**	**Gates**	**Depth**
Multiplication:		n	0	$n+m$	n^2+nm	$n+m$
Estimation:		m	0	$n+\{m,...,1\}$	$nm+(m^2+m)/2$	$2n+2m$
Correction	QFT†:	0	1	$n+1$	$(n^2+n)/2$	
	QFT:	0	1	n	$(n^2-n)/2$	$2n$
	Add N:	1	0	n	n	
Output transform:		0	1	n	$(n^2-n)/2$	$2n^\ddagger$
Uncompute	QFT†:	0	1	$m+1$	$(m^2+m)/2$	
	$t \to 0$:	n	0	m	nm	

‡Uncomputation steps executed in parallel with output QFT†

6.3 Quantum Fourier Barrett reduction

Like Montgomery reduction, the benefit of Barrett reduction in the classical case is in the replacement of division with integer multiplication (Algorithm 2). The quantum Barrett reduction procedure is therefore similarly well-suited for arithmetic in the quantum Fourier basis.

As in the division and Montgomery reduction based multiplication procedures, we begin with Fourier calculation of $|t\rangle^{\Phi}_{n+m}$ (Algorithm 2, step 1). In the Barrett case, we also accumulate the approximate product $\widetilde{Xy}$ (step 2) with simultaneous Fourier-basis adders. The approximate product is then used to compute $\tilde{q}$ (step 3), requiring its transformation to binary representation prior its Fourier-basis multiplication by μ. Similarly, $\tilde{q}$ is used as a quantum multiplicand in steps 4, 9, and 13, and so must be transformed to its binary representation prior to use in these steps.

The quantum Barrett reduction procedure requires two comparison operations (Algorithm 2, step 5 and step 10), requiring the usual QFT† and QFT operations in order to extract sign bits. However, while the first requires full-register transformations, the second comparison is limited to just the most significant bits of the accumulation register (the number of bits required is equal to the size of $\tilde{q}$). After a full-register QFT† to extract the sign, we therefore only need to transform these MSBs of the register to the Fourier basis for the re-addition of $\widetilde{Xy}$ and subtraction of qN (steps 10 and 13). We finally transform these bits back to their binary representation so that both the output and input states of the modular multiplier are binary.

The final uncomputation of $\tilde{q}$ and $\widetilde{Xy}$ (steps 14 and 15) requires the reversal of their initial computations and transformations. As in the uncomputation stages of the Φ-DIV and Φ-REDC operations, the Fourier uncomputation of $\widetilde{Xy}$ is requires subtractions controlled by each bit of the input state, and therefore maintains the initial multiplication's depth of n controlled rotations. Combined with the three full-register QFTs required for comparisons, the overall Fourier Barrett multiplication operation has a depth of $7n + \mathcal{O}(m)$ controlled rotation gates, identical in leading order to the Montgomery reduction operator. As with both Φ-DIV and Φ-REDC, if we bound the precision of the component QFTs, the total leading-order gate count is just that of the initial multiplier.

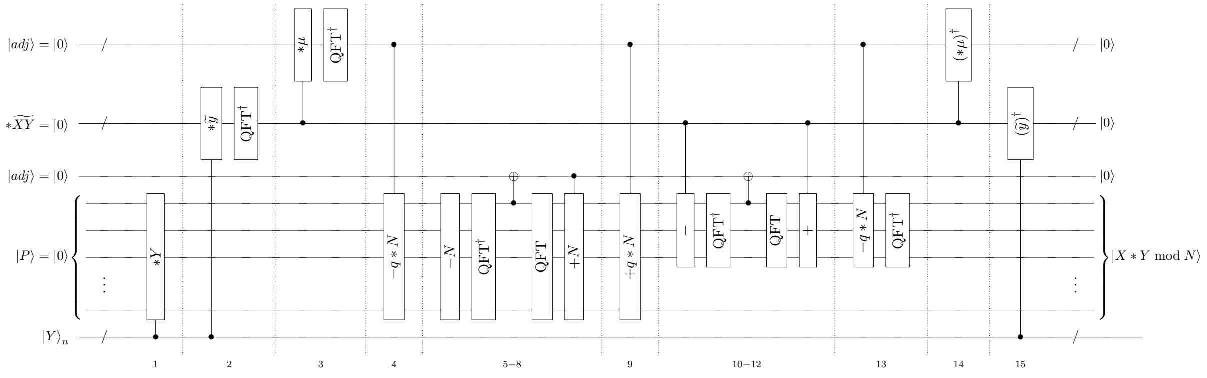


Figure 12: Barrett multiplication circuit using Fourier arithmetic. The numbers in the figure correspond to the steps of Algorithm 2.

7 Resource Evaluation

In this section we present a detailed resource analysis of the modular multiplier designs introduced in the previous sections, alongside that of the standard modular-adder approach (Section 2). We focus our analysis on the quantum-classical modular multiplier (where one input is a classical parameter); as described in Section 2.5, in-place multiplication with two quantum inputs would require a circuit to calculate the multiplicative inverse of one of the inputs, and this inverse circuit would dominate the resources of the multiplier. If the particular details of this analysis are not of interest, a summary important takeaways can be found in Section 7.5.

We explicitly generate circuits for each multiplier design with various sizes, different classical constants, and different classical moduli. We utilize the four adders discussed in the appendix: the Fourier basis adder [7], the logarithmic-depth prefix (or carry-lookahead) adder [13], the linear-depth majority ripple adder [12], and the rippled prefix adder defined in Section A.1 of the appendix. Resource analysis is performed assuming two different hardware models: the first treating all gates equally, and the second accounting for the relative complexity of each gate in the context of a fault-tolerant logical circuit using quantum error-correcting codes. With each model, we determine an overall circuit *size*, or combined cost of all gates, and *depth*, or total latency of the circuit allowing for unbounded parallelization. We do not include locality constraints in our hardware models. The impact of locality is primarily dependent on the particular architecture and the addition technique employed, with roughly the same impact on each multiplier design.

7.1 Evaluation Methodology

For each evaluation, we construct a full quantum circuit consisting of gates from a set natural to the adder that is being used. For example, the Fourier basis adders predominantly require controlled-Y rotation gates, while all circuits utilize X gates, with zero, one, or two controls. We use a C++ program to generate each circuit. The program can generate sequences of arbitrary size, to a practical limit. The user specifies the classical modulus and multiplicand, which are folded into the circuit to produce an optimized circuit unique to those classical inputs. Circuit depth for each hardware model is determined by then running the generated circuit through a scheduler, which places gates into parallel time slots based on the specified latency of each gate in that model. The scheduler reorders gates according to standard commutation rules: gates that do not share any qubits can be trivially reordered, and two gates that do share qubits can be reordered if they both commute either with the Pauli Z operator or the Pauli X operator. Finally, each circuit is verified for a variety of input values with a separate gate-level simulation program.

Our resource evaluations assume two different hardware models. In the first model (which we will label "equal-latency"), all gates, including controlled arbitrary angled rotation gates, have the same latency.

	Hardware Model	
Gate	Equal-latency	Fault-tolerant
CNOT	1	1
H	1	1
T	1	10
Toffoli	1	40
Pauli	1	1
$\mathrm{Y}(\alpha)$	1	$66\log_2(n) + 33$
$\mathrm{CY}(\alpha)$	1	$\sim 66\log_2(n) + 35$

Table 6: Unit costs of gates for two hardware models. The first "Equal latency" model assumes the same latency for all gates. The second, "Fault-tolerant" model enforces gate-dependent latencies, representing a rough cost in primitive gates available to an error-correcting code. See the text for a description of the cost of the single-qubit $\mathrm{Y}(\alpha)$ and controlled $\mathrm{CY}(\alpha)$ gates.

This provides a useful characterization of our circuits, and comparison to other circuits in the literature. However, it is not realistic for architectures that implement limited gate sets. The second model ("fault-tolerant") is motivated by the gates that are typically available on logical qubits implemented using a quantum error correction code. For standard Calderbank-Steane-Shor (CSS) codes [31], Clifford gates such as CNOT and H are easy to implement whereas gates such as T and Toffoli require the use of more complicated circuitry and potentially the use of magic-state distillation, which can be costly. For this reason we assign a cost of 10 to the T gate and a cost of 40 to the Toffoli gate.

In order to construct controlled rotation gates fault-tolerantly, we decompose each into the sequence of two CNOT gates and two unconditional rotations depicted in Figure 13. As shown, the unconditional $\mathrm{Y}(\alpha/2)$ rotation in the decomposed gate commutes with the remainder of the operation, as well as adjacent single-qubit and controlled rotations targeting that qubit. We can therefore collect and merge these commuting gates into a single unconditional rotation per qubit, as shown in Figure 14. We then expect that, in the fault-tolerant model, each decomposed controlled rotation has an amortized cost of one rotation gate in addition to the two CNOT gates. Also shown in Figure 14, the control qubit of the decomposed gate is only involved in the CNOT operations; given the low latency of CNOTs relative to rotation gates in this model, the latter can be be further parallelized between the CNOTs of adjacent gates sharing a control qubit.

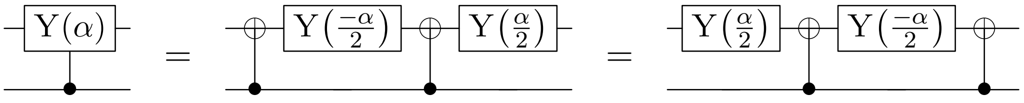


Figure 13: Decomposition of controlled rotation gate into CNOTs and single-qubit rotations. Because the control qubit is not active during the single-qubit rotations, this decomposition also allows for greater parallelization than a composite two-qubit gate of the same latency.

Arbitrary single-qubit rotations cannot be implemented directly on CSS codes, and instead must be approximated with sequences from the finite set of available gates. A sequence of T and H gates approximating a single-qubit rotation to an accuracy ϵ can be constructed with $3\log_2(1/\epsilon)$ steps, where each step contains an H gate and a T gate raised to an integer power [32]. Noting that the complexity of implementing a T^k gate on a CSS code should be no more than that of a single T gate, we assume that the cost of each (HT^k) step is the same as that of one H gate and one T gate.

The necessary accuracy ϵ for fault-tolerance is a function of the total number of rotation gates in the multiplier. To first order, the Barrett and Montgomery multipliers require $2n^2$ controlled-Y rotations, each of which can be implemented using two CNOT gates and one effective single qubit rotation. We therefore choose $\epsilon = 1/(2n^2)$, requiring $3\log_2(2n^2) = 6\log_2(n) + 3$ steps to approximate the single-qubit rotation. Incorporating the costs of T and H gates determined above, we estimate a total cost of

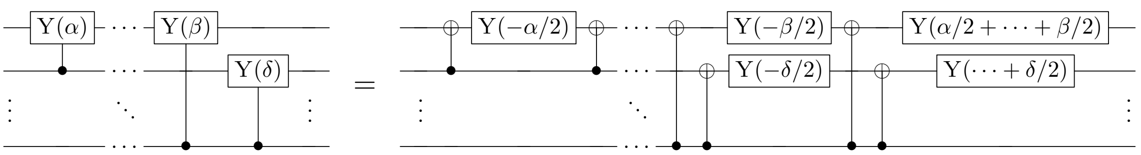


Figure 14: The outer rotation gate of the decomposed controlled-Y rotations (Figure 13) can be commuted through adjacent gates, and combined into a single rotation per qubit. This freed control qubit enables further parallelization of adjacent gates, taking advantage of the low latency of the CNOT gate relative to the rotation.

$66\log_2(n) + 33$ per rotation, or $66\log_2(n) + 35$ for the controlled-Y rotation gate in Figure 13. The cost of each gate in the two hardware models is summarized in Table (6).

Given the finite precision with which we are approximating rotation gates, it is also reasonable to explicitly limit the precision of the controlled rotation gates we can implement. In particular, we can drop rotations with an angle $\theta^2 \approx \epsilon$, as these are sufficiently well approximated by the identity operation. This simplification can dramatically reduce the overall gate count, while in the presence of decoherence possibly improving operator fidelity [23]. For example, the set of rotation gates required by the QFT is described by $\{\theta = \pi/2^k \mid k = 1, ...n\}$, and so can be truncated after $k \sim \log n + 2$ as described in [23]. We use this approximation in the analysis of all Fourier-basis circuits.

For each analysis, we construct sequences with increasing input register widths $8 \leq n \leq 2048$. To avoid pathological cases, for a given n we generate and average the behavior of eight unique quantum circuits, with (when applicable) classical moduli $2^{n-1} < N < 2^n$ and multiplicands $0 < X < N$ randomly selected from uniform distributions. We expect a slight reduction in circuit size from the worst case due to this randomization; for example, each binary adder contains a number of gates conditioned on the bits of the classical addend, half of which can be expected to be eliminated for a randomly chosen addend (with negligible variance for large n).

7.2 Adder Comparison

We first evaluate the different quantum addition techniques we will use to benchmark our modular multipliers. In Figure 15, we show the results of simulations of each of the adders using the two hardware models described above. Each circuit is generated to perform in-place, controlled quantum-classical addition, with a quantum input register size $0 < n \leq 2048$ randomly chosen from a logarithmic distribution. In the equal-latency model (i.e. assuming all gates are equivalent), we see the expected logarithmic scaling of the prefix adder depth, and linear scaling of the ripple-carry and prefix-ripple constructions. As in [12], asymptotic latency of the ripple-carry adder is $\sim 2.0n$. The prefix-ripple circuit improves upon the ripple-carry, approaching a depth of $\sim 1.0n$ in our data set, but doubles the total number of gates required. This observed depth matches what we would expect for a circuit dominated by one Toffoli gate per two bits (where both the forward and reverse circuits required for the in-place operation contribute $n/2$). The Fourier-basis adder simply consists of n controlled rotation gates in this model, and afford no parallelization due to the due to the shared control qubit.

The latencies of the three binary adders are dominated by Toffoli gates, and so are increased proportionally in the fault-tolerant model. However, the total gate count of the prefix adder, having a greater overall proportion of Toffoli gates, is increased more significantly than the ripple and prefix-ripple circuits. The decomposition of rotation gates required in the case of Fourier-basis addition in this model results in $\mathcal{O}(n \log n)$ total gates, dominating the linear sizes of the binary adders. However, the latency of the fault-tolerant Fourier-basis adder is only increased by a factor of two at large n. As described above, the logarithmic-depth single-qubit rotations can be parallelized such that the asymptotic latency is dominated by the $2n$ CNOT gates sharing a control qubit. Below $n < 652$, the latency of two single qubit rotations is greater than $2n$ and so must dominate the circuit depth. This transition is clearly visible in the Fault-tolerant latency plot of Figure 15. The depth of the Fourier-basis adder is comparable to the logarithmic-depth prefix adder prior to the dominance of the $2n$ CNOTs, and consistently below that of the ripple-carry and prefix-ripple adders.

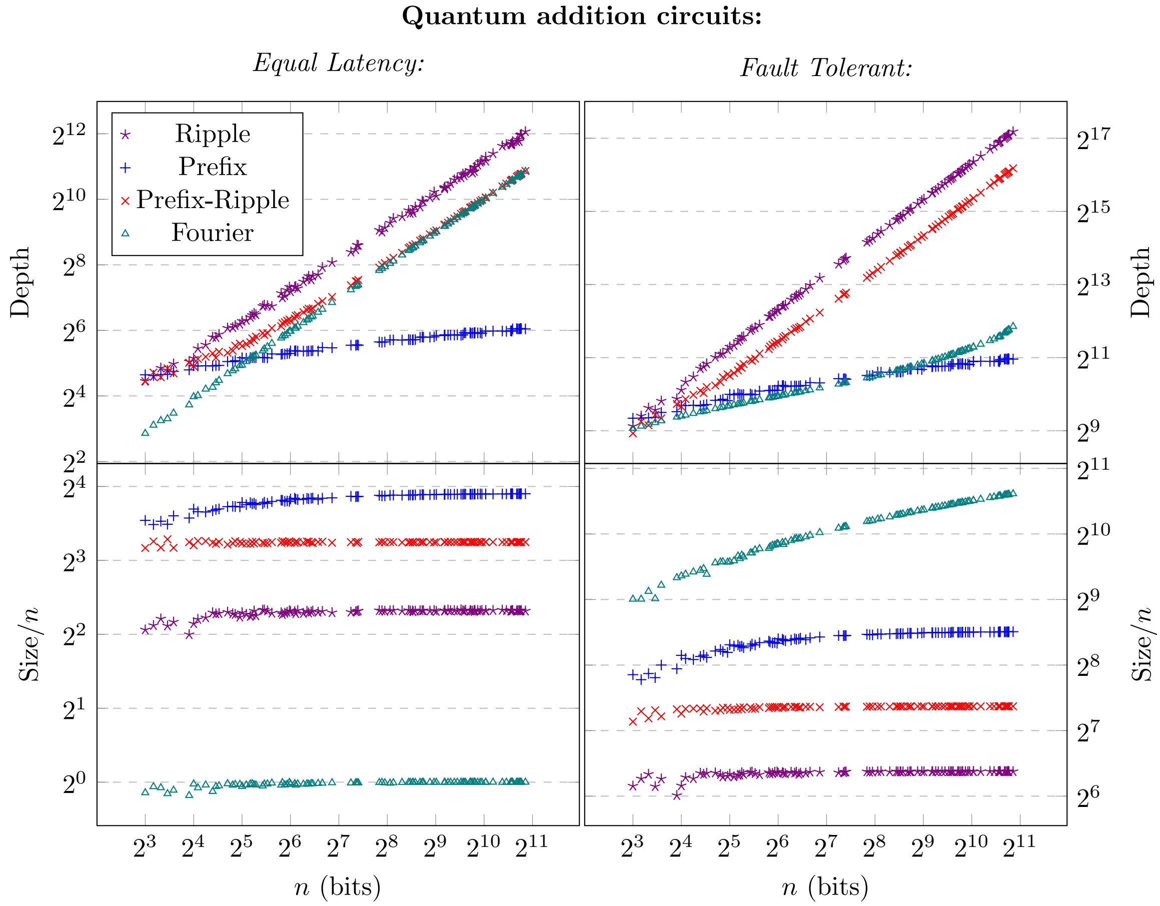


Figure 15: Resources required for standalone quantum adder implementations. The details of the adders are described in the Appendix. The depth is the latency of the parallelized circuit and the size is the total number of gates in the circuit. The inflection point in the depth of the fault-tolerant Fourier-basis adder is where the $2n$ CNOT gates from the adder's control qubit begin to dominate the logarithmic-depth fault-tolerant rotations. Logarithmic depth could be preserved by fanning out the control, at the cost of an additional $\sim n$-qubit register, but this is unnecessary in any of our constructions.

7.3 Modular Multiplier Components

The quantum division, Montgomery multiplication, and Barrett reduction circuits introduced in the preceding sections share a common high-level three-stage structure, diagrammed in Figure 16. The initial Multiplication stage, identical in the three designs, consists of n controlled adders arranged as an $(n + \log_2 n)$-qubit quantum accumulator. The goal of each design has been to reduce the overall complexity of quantum modular multiplication to that of this step. We therefore begin by generating circuits for the Multiplication operation in isolation with each of the quantum addition techniques outlined above. The sizes and depths of the generated circuits are displayed in Figure 17. Observing these results in comparison to Figure 15, in each case the total gate count predictably approaches that of n isolated adders. When constructed with any of three binary addition circuits, the overall circuit latency of the multiplier is increased by the same factor, indicating that the scheduler did not find significant

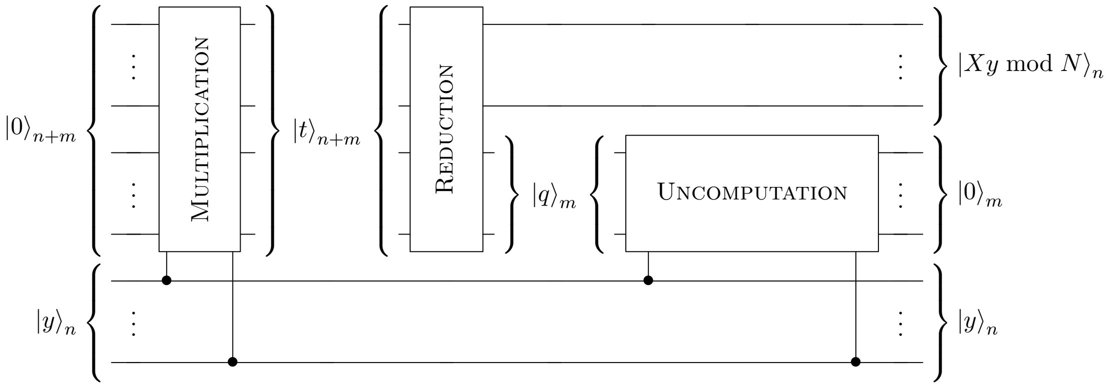


Figure 16: Generalized three-stage structure of our three proposed out-of-place modular multipliers, where $m \approx \log_2 n$. The size of each circuit is asymptotically dominated by the initial n-addition Multiplication stage, with the Reduction stage requiring only $\mathcal{O}(\log n)$ adders, and the Uncomputation stage adders only spanning $\mathcal{O}(\log n)$ qubits. As in Section 2.2, the in-place multiplier then requires the structure to be repeated in reverse.

opportunity to parallelize gates between adjacent adders. The commutable rotations composing the Fourier-basis multiplier, however, enable parallelization across the entire accumulator. In the equal latency model, this allows n simultaneous controlled rotation gates in each time step, circumventing the control bottleneck of the single adder and reducing the total latency to $(n + \log_2 n)$. In the fault-tolerant model, the sequential adders further allow the commutation and combination of single-qubit rotations, as in Figure 14. Accordingly, the circuit latency observed in Figure 17 is close to $66n \log_2(n)$, or that of n single-qubit rotations alone. Even after this incurred cost in the Fault-Tolerant model, the Fourier-basis Multiplication circuit has the lowest depth among the adders. Unfortunately, the total gate count in this model is $\mathcal{O}(n^2 \log n)$, asymptotically dominating the $\mathcal{O}(n^2)$ gates required by the three binary circuits.

The division, Montgomery, and Barrett circuits are differ in their respective Reduction stages, in which the modular reduction (or Montgomery reduction) is computed from the non-modular product, alongside an $\mathcal{O}(\log n)$-qubit quotient-like term (which is necessarily produced by a generic information-preserving modular reduction). The key to the performance of these circuits is in limiting the number of additions required in this stage to $\mathcal{O}(\log n)$. The resources required for the Reduction stage of each proposed multiplier, in both the Fourier-basis and binary (prefix) implementations, are compared in Figure 18. Due to its logarithmic dependency on register size, a small number of adds relative to the accumulator size makes the decreased latency of the prefix adder relative to the other binary circuits even more pronounced in this stage, allowing the reduction to be performed in poly-logarithmic time compared to the $\mathcal{O}(n \log n)$ depth achieved with ripple and prefix-ripple adders. For each binary adder, the circuit size and latency of this stage are overwhelmingly dominated by the initial Multiplication stage in Figure 17.

Conversely, as described in Section 6, the need for at least one comparison operation within each multiplier's Reduction stage necessitates the invocation of intermediary QFTs in a Fourier-basis implementation, making this the highest-latency stage of the Fourier-basis modular multipliers. The Barrett and Montgomery circuits, requiring a constant number of QFTs, have linear depth in the equal-latency model. The $\mathcal{O}(\log n)$ comparisons in the Φ-DIV circuit increase this latency to $\mathcal{O}(n \log n)$, asymptotically dominating the linear depth of the Multiplication stage. However, assuming finite-precision rotations, the gate counts of the circuits are $\mathcal{O}(n \log n)$ and $\mathcal{O}(n \log^2 n)$, respectively, all scaling comparably to the binary Reduction circuits and asymptotically smaller than the Fourier-basis Multiplication stage.

The final step of each modular multiplier is a $(\log n)$-bit quantum accumulator, consisting of n adders controlled by the bits of the input state $|x\rangle_n$. The Uncomputation step in Figure 16 serves

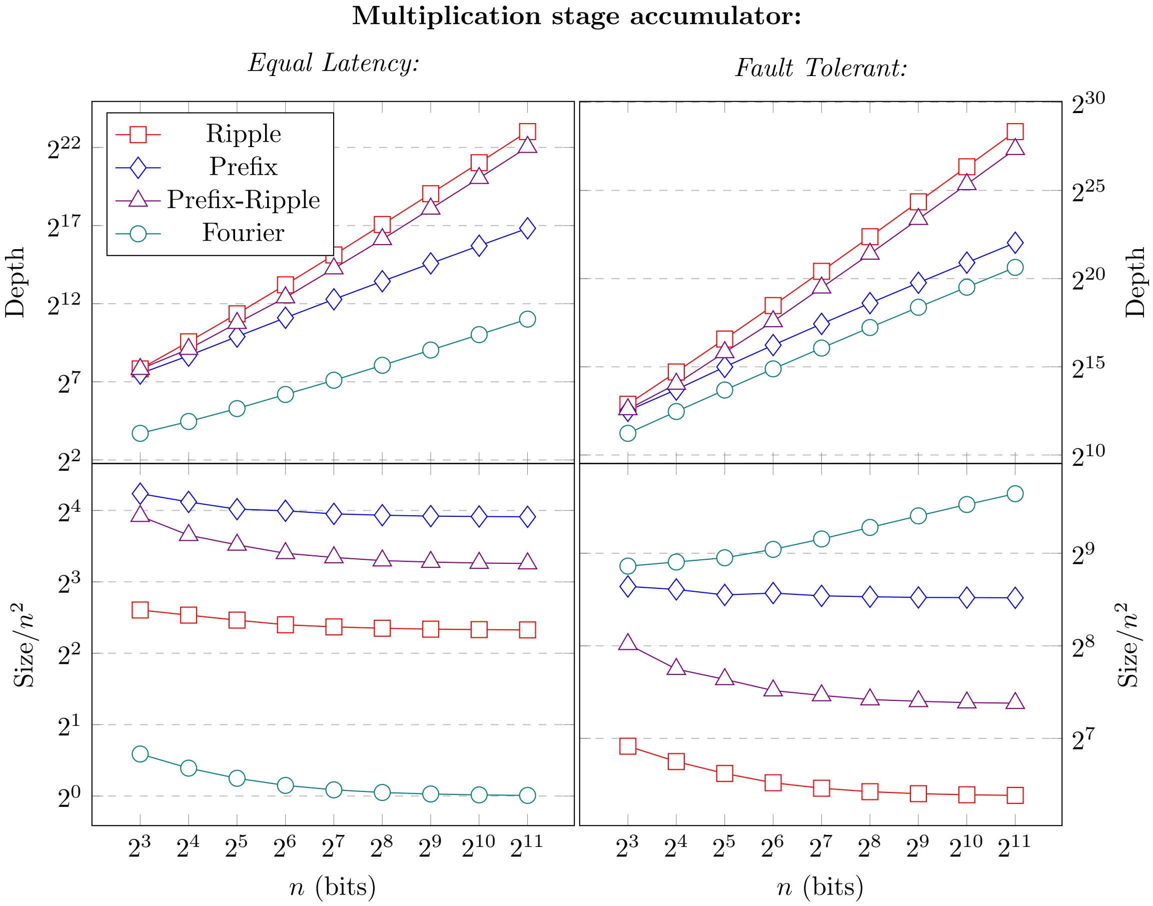


Figure 17: Resource requirements of the initial Multiplication circuit common to the three modular multipliers, comprising n sequential $(n + \log_2 n)$-qubit quantum adders. In both hardware models, we see the expected asymptotic speedup of the logarithmic-depth prefix adder over the linear-depth ripple and prefix-ripple circuits (upper plots), and the corresponding increase in circuit size (lower plots). In the equal-latency model, the depth of the parallelized Fourier-basis multiplier is linear in n, while the prefix adder scales as $\mathcal{O}(n \log n)$ and the ripple and prefix ripple as $\mathcal{O}(n^2)$. In the Fault-tolerant model, the $\mathcal{O}(\log n)$ depth of controlled rotation gates results in the higher $\mathcal{O}(n^2 \log n)$ asymptotic scaling of the Fourier-basis multiplication, while its parallelized depth remains the least of the four.

to uncompute the $(\log_2 n)$-qubit quotient-like term produced by the preceding Reduction stage. As described in Section 3.3, with the binary circuits we can parallelize adders over work qubits already necessary for the preceding stages, so as to match the poly-logarithmic circuit depth achieved with prefix adders in the Reduction stage. The resource requirements for the parallelized Uncomputation stage constructed with each addition method are shown in Figure 19. In the case of the binary adders, we immediately find that the size and depth of this step is then asymptotically insignificant in comparison to the initial Multiplication stage (Figure 17). In the equal-latency model, we find that the ripple-carry circuit has by a small margin the lowest latency for $n < 2^6$, at which point it becomes comparable to the prefix-ripple circuit. The latter performs marginally better in the fault-tolerant model. In all cases, the prefix circuit has the greatest size and latency of the binary adders. Observing Figure 15, in the range of n being analyzed, we expect the depth of a $(\log_2 n)$-qubit adder to be comparable with any of the three circuits, while the additional qubits required by the prefix adder reduces the number of additions we can perform simultaneously in the parallelized accumulator. We expect that for even larger

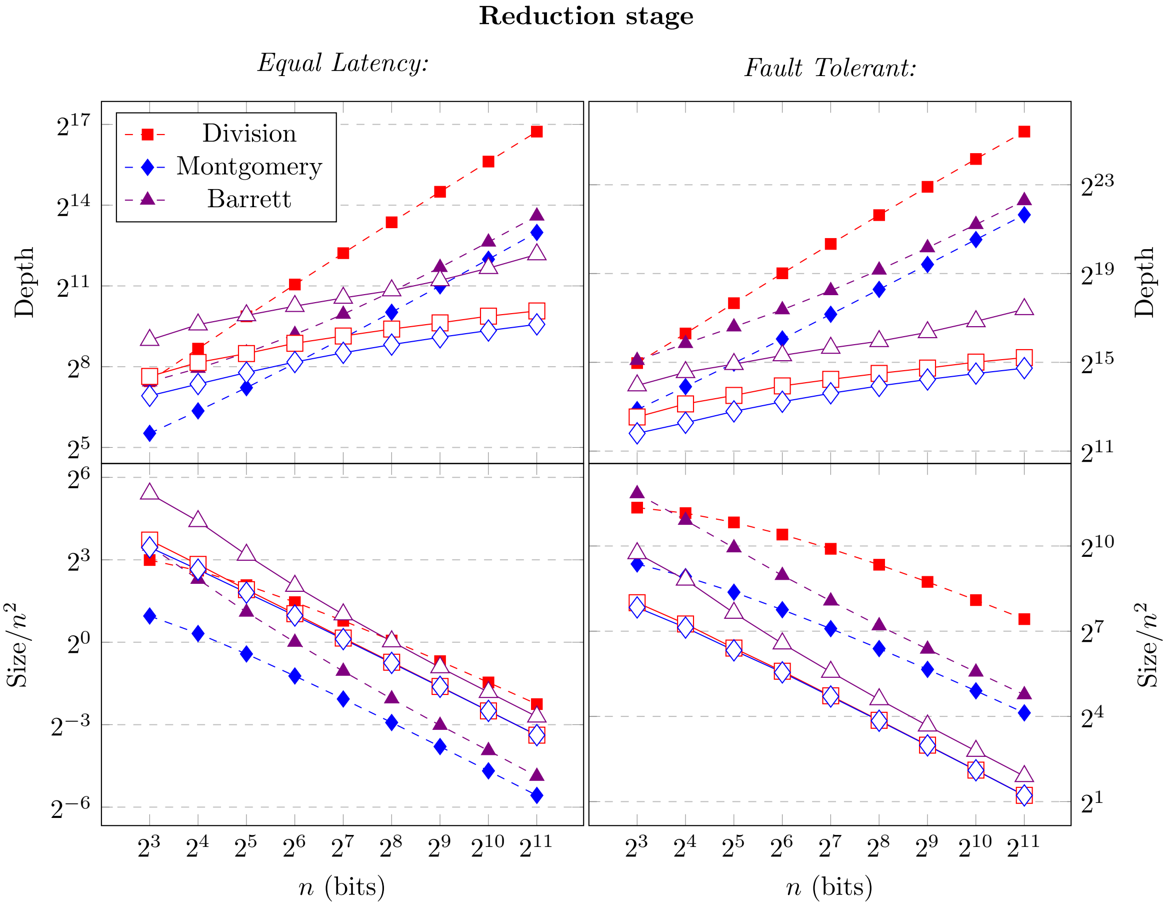


Figure 18: Reduction stage of proposed modular multipliers, constructed with prefix (hollow marks) and Fourier-basis (solid marks) adders. In the prefix case, the circuit size and latency of each circuit is comparable to the Uncomputation stage and asymptotically dominated by the initial Multiplication. The Fourier-basis Reduction stages also require asymptotically fewer gates than the corresponding Multiplication stages, but have greater depth due to the remaining of comparison operations. Like the Multiplication stage, the Barrett and Montgomery circuits have linear depth in the equal-latency model, whereas the division circuit requires $\mathcal{O}(\log n)$ comparisons and therefore has $\mathcal{O}(n \log n)$ depth.

multiplier sizes the prefix adder would again provide the fastest circuit. In the Fourier-basis case, we cannot further parallelize adders, resulting in a linear depth in the equal-latency model and $\mathcal{O}(n \log n)$ depth in the fault tolerant model. In both models, the gate count of the Fourier-basis Uncomputation remains asymptotically dominated by the Multiplication stage.

7.4 Multiplier Comparison

We now benchmark the optimized resource requirements of each of the quantum modular multipliers, constructed with both binary and Fourier-basis adders. For each design, we construct circuits for in-place modular multiplication, incorporating quantum control with the third method outlined in Section 2.3 (at the cost of $3n$ controlled-swap operations, decomposed into Toffoli and CNOT gates). We begin with the binary-basis multipliers. Given the results of the previous section, we use the prefix adder for all $\mathcal{O}(n)$-qubit quantum adders, and the prefix-ripple adder for the $(\log_2 n)$-qubit Uncomputation-stage accumulator. The results for each multiplier, for both hardware models, are shown in Figure 20, where

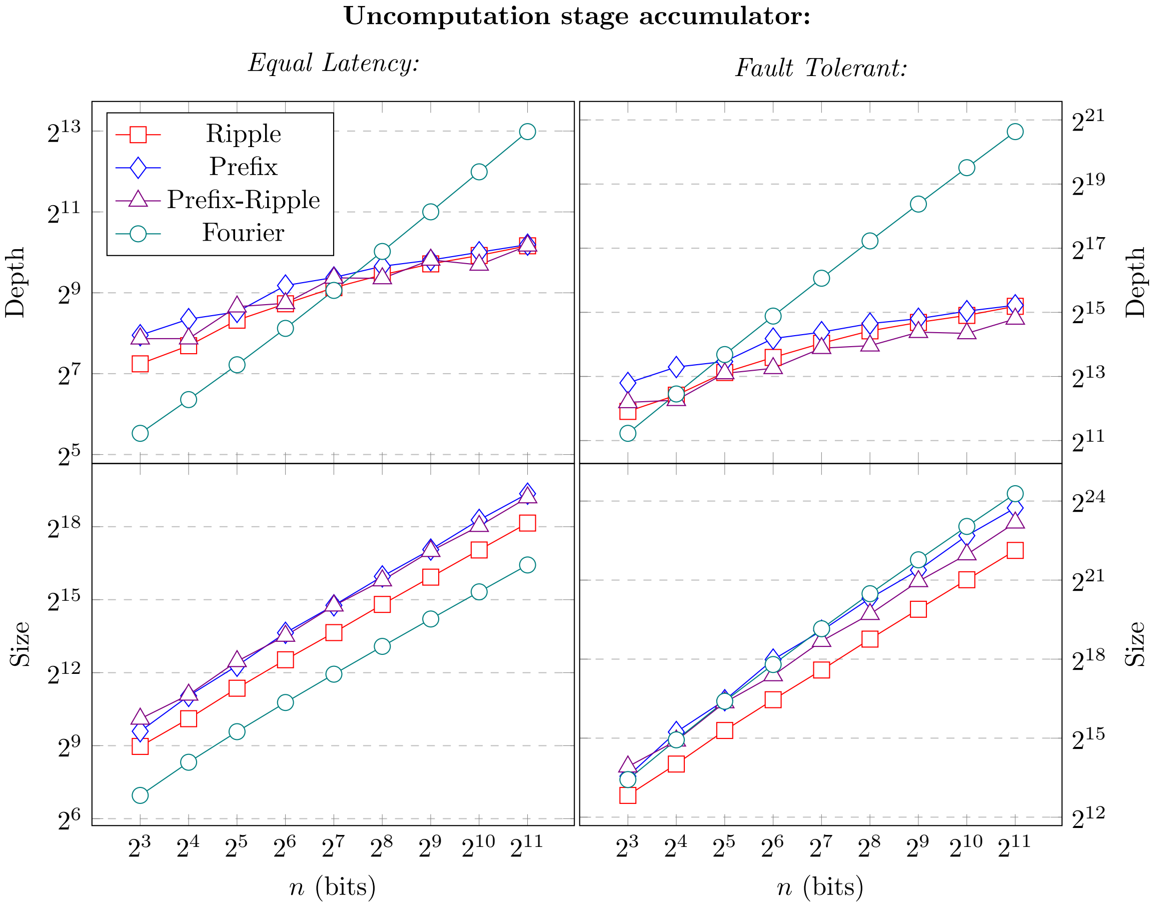


Figure 19: Uncomputation stage accumulator: each of the proposed modular multipliers requires the clearing of a $(\log_2 n)$-qubit quotient register, requiring additions conditioned on each bit of the n-qubit input register. Using binary adders, these can either be executed sequentially or parallelized over the work qubits necessary for the preceding stages. The Fourier-basis implementation does not afford this parallelization, resulting in its linear depth in the equal-latency model.

we have normalized by the expected asymptotic scaling ($\mathcal{O}(n \log_2 n)$ depth and $\mathcal{O}(n^2)$ size). As promised, we find that the asymptotic latency and gate count of each of the three modular multipliers proposed in this work is twice that of the initial Multiplication stage alone (where the factor of two results from the two out-of-place circuits required for in-place modular multiplication). Circuits generated with standard modular addition approach, also shown in Figure 20, increase both the size and depth by another factor of three, corresponding to the three quantum additions required for each modular adder.

In Figure 22, we show the resources consumed by the same multipliers constructed with Fourier basis operations. Here, the circuit depths of the different modular multipliers deviate significantly, driven by the intermediary QFTs necessary for comparison or modular reduction. The standard modular-addition technique, as introduced in [8], requires a total of $8n$ QFTs, resulting in the $\mathcal{O}(n^2)$ depth observed in the equal-latency model in Figure 22. The Reduction stage of the division-based circuit drives its $\mathcal{O}(n \log n)$ depth, while both the Barrett and Montgomery circuits display linear depth at large n, plateauing below the $14n$ worst-case latency determined in Section 4. At large n, the total number of gates required by all three proposed circuits coalesce to about $2n$. This asymptotic size is twice that observed in Figure 17, indicating that all three circuits are dominated in size by their non-modular multiplication operations. As a result of the imposed finite precision of Fourier rotations, the number of

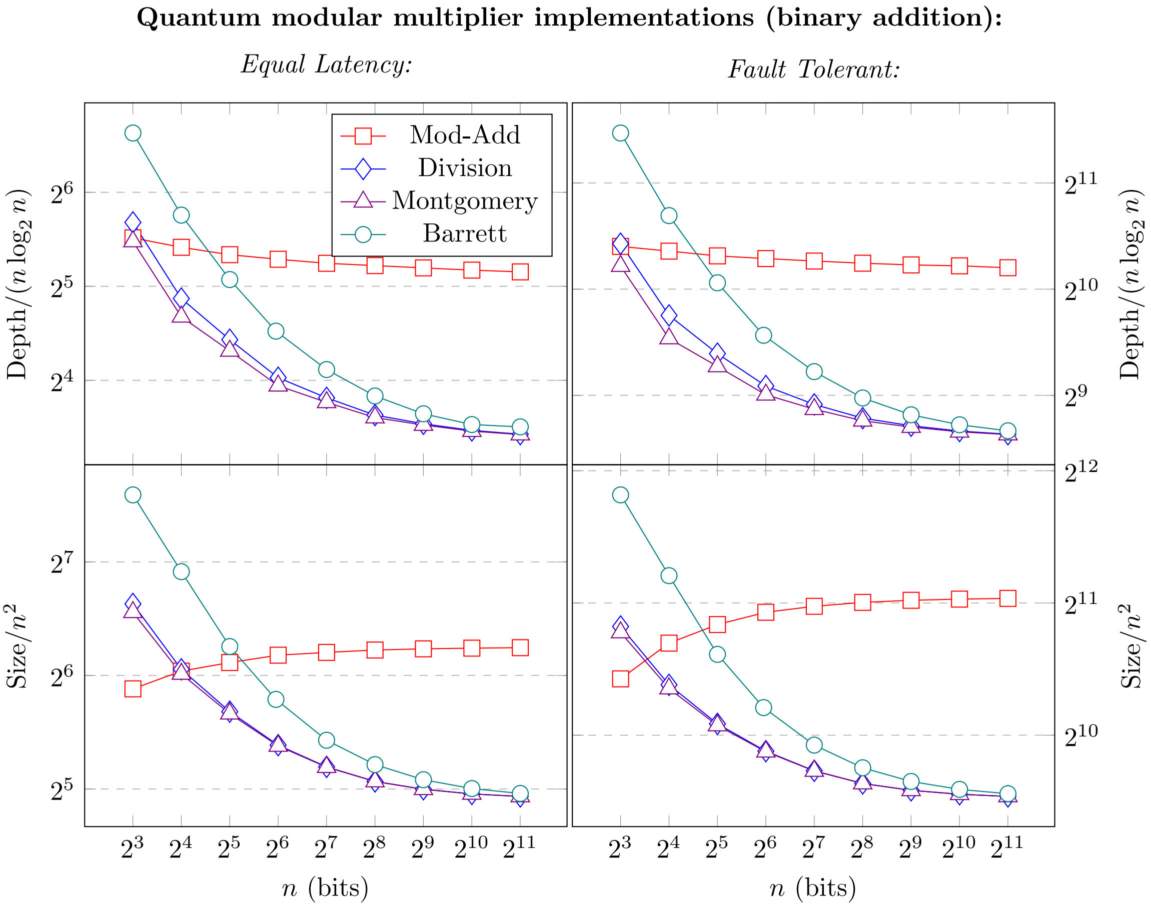


Figure 20: In-place, controlled modular multipliers constructed with binary quantum addition circuits. We use the logarithmic-depth prefix circuit for the $\mathcal{O}(n)$-qubit adders, but the prefix-ripple circuit for the $(\log_2 n)$-qubit UNCOMPUTATION-stage accumulator. Both size and depth are normalized by their expected asymptotic scaling.

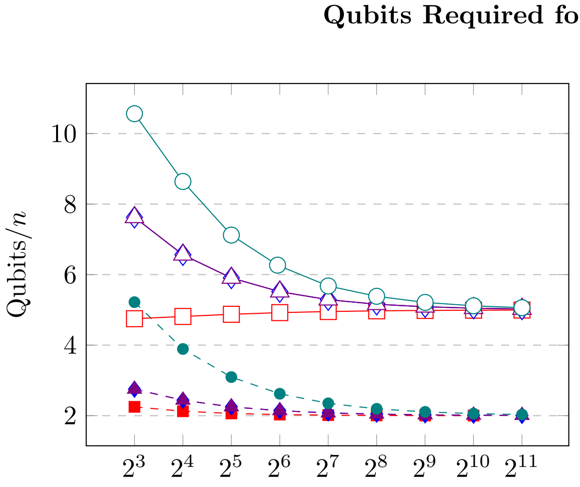
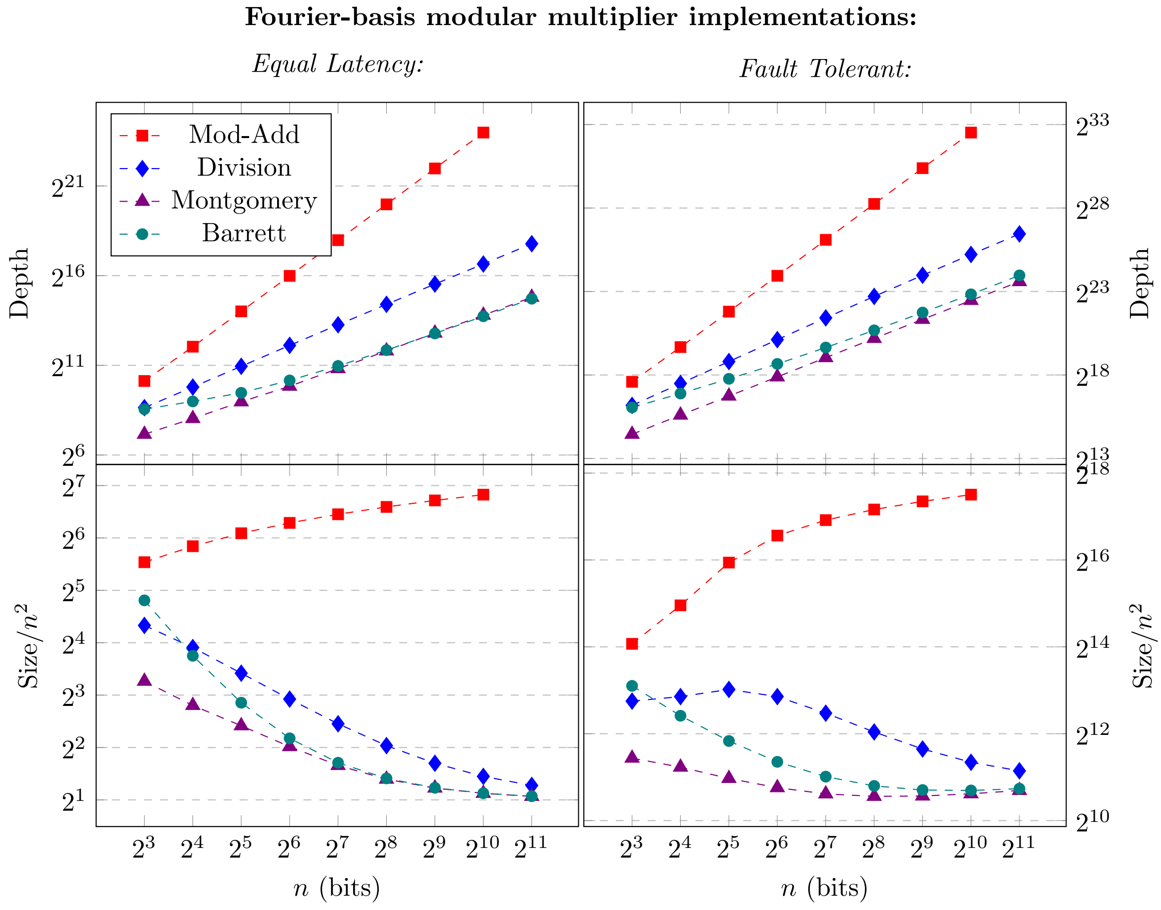


Figure 21: Qubits cost of modular multipliers with prefix (hollow marks) or Fourier-basis (solid marks) addition. Asymptotically, each proposed circuit requires $5n$ qubits with prefix adders, identically to the standard modular-addition approach. The Fourier circuits require just $2n$ qubits asymptotically, matching the low-ancilla but much more costly circuit in [8]. At small n, the additional $\mathcal{O}(\log n)$ qubits required by our circuits become apparent. The Barrett multiplier further requires an $\mathcal{O}(\log n)$-qubit register for $|\widetilde{Xy}\rangle$, causing its dominance at low n.

gates required by the division algorithm's Reduction stage is eventually dominated by the n^2 gates in the initial Multiplication stage (however, as seen in Figure 22, the size of the division-based circuit remains greater in the range of circuits observed).

In the fault-tolerant model, the total gate count of each circuit is increased by a factor of $\sim 66\log_2(n)$ factor, corresponding to the cost of decomposing rotation gates. However, after scheduling, we find that the total latency is only increased by $\sim 40\log_2(n)$, demonstrating the additional parallelization enabled by the decomposition of controlled rotation gates described in Section 7.1.


Fourier-basis modular multiplier implementations:

Equal Latency:

Fault Tolerant:

Figure 22: In-place, controlled modular multipliers constructed with Fourier-basis adders. Both size and depth are normalized by their expected asymptotic scaling.

Finally, we plot the prefix and Fourier-basis implementations of the three proposed multipliers together in Figure 23, and their corresponding qubit costs in Figure 21. A principal motivator for Fourier-basis quantum arithmetic is in reducing the number of required qubits. Motivating its introduction in [8], the Fourier-basis modular-addition-based circuit requires at most $2n+3$ qubits for any n. The number of qubits required for Fourier-basis implementation of each of our proposed circuits also approaches $2n$ asymptotically, requiring only a logarithmic increase in ancilla. Notably, prior speedups to Fourier-basis modular multiplication have come at the expense of additional $\mathcal{O}(n)$-qubit registers [5, 14]. Comparatively, the ripple and prefix-ripple circuits each consume $\sim 3n$ qubits, requiring an additional n-bit register to contain carry bits. The prefix circuit, while having the lowest latency, of the binary adders requires $5n$ qubits. In each case, the asymptotic ancilla cost for modular multiplication is that of n-qubit addition with the chosen adder.

While the $2n^2$ gate count of and linear depth of the Fourier-basis Barrett and Montgomery multipli-

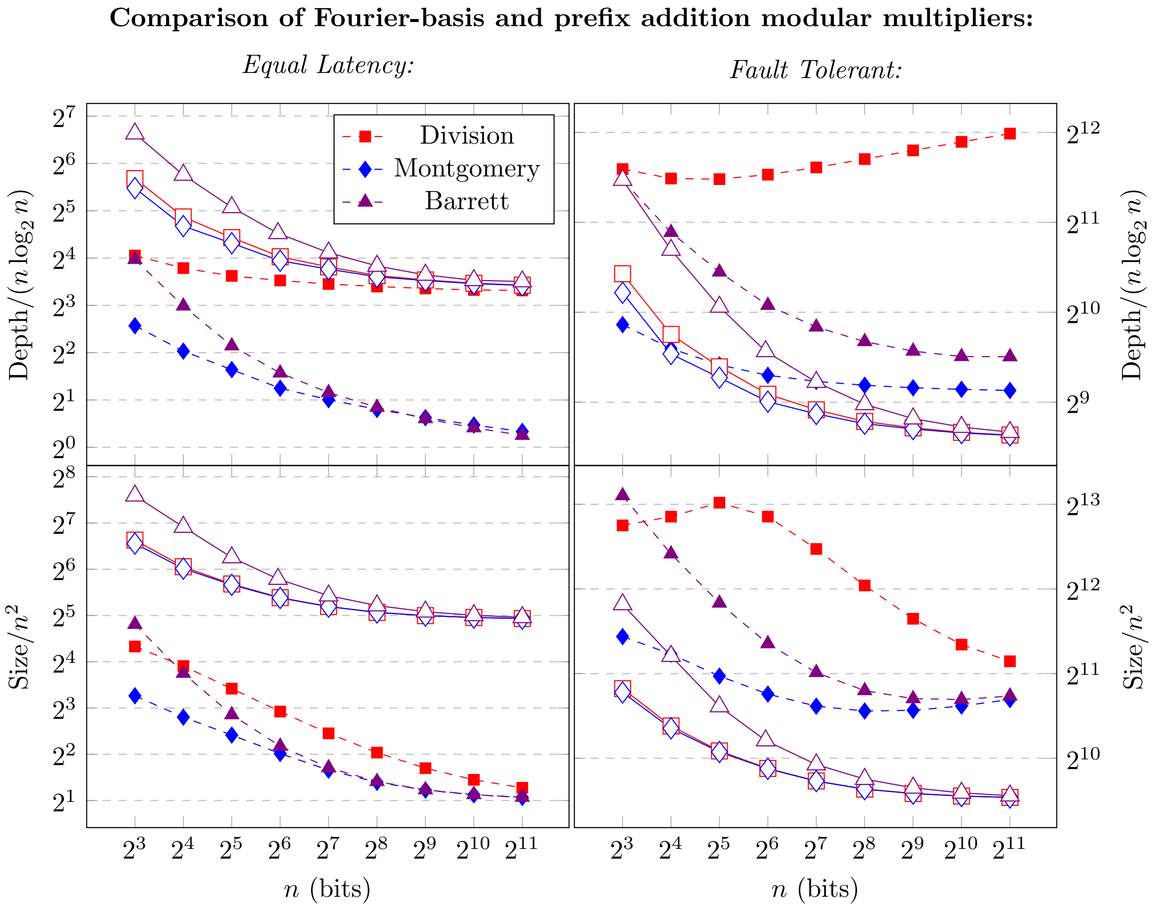


Figure 23: Proposed modular multipliers constructed with both Fourier-basis arithmetic (solid marks) and prefix addition (hollow marks).

cation circuits in the equal latency model clearly outperform the $\sim 2^5n^2$ gates and $\mathcal{O}(n \log n)$ latency observed of the multipliers constructed from prefix adders, this comparison is likely not valid in conjunction with quantum error correction. In the more realistic fault-tolerant model, the Barrett and Montgomery circuits implemented with Fourier-basis addition and all three proposed multipliers constructed with prefix adders have $\mathcal{O}(n \log n)$ asymptotic latency. In our simulations, the latencies of the Fourier-basis circuits were about 75% greater than those from the prefix-adder circuits, and fall below the latencies of circuits constructed in the typical modular-addition approach with either adder. The small latency increase indicates a potentially reasonable tradeoff for the qubit reduction enabled by the Fourier-basis circuits, given reasonable architectural assumptions. Comparatively, the latencies of the modular addition circuits are increased from $\mathcal{O}(n \log n)$ to $\mathcal{O}(n^2 \log n)$ when implemented with fault-tolerant Fourier-basis arithmetic instead of prefix adders. Further, while the gate counts of all of the Fourier-basis circuits grow faster than the binary circuits by a factor of $\mathcal{O}(\log n)$, the discrepancy remains within a factor of three in the large range of n observed.

7.5 Summary

We find a number important takeaways from the above experimental analysis of the quantum division, Barrett reduction, and Montgomery reduction circuits for quantum modular multiplication. First, empirical analysis confirms the dominance of the non-modular multiplication process in the complexity of the three modular multiplication procedures introduced in this work. Benchmarking the individual stages of the modular multipliers with a variety of binary quantum adders, we have found that the gate count and circuit latency of the modular reduction and uncomputation components become insignificant at large n compared to the initial multiplication step of each circuit. Additionally, the observation of the different addition circuits further enabled the characterization of multiplier stages in relation to their component adders, as well as corresponding optimizations. For example, in the range of register sizes considered, the normally-fast prefix adder turns out to perform the worst of the adders in the case of the width-sensitive parallelized Uncomputation accumulator. For the combined, in-place modular multiplier, the total asymptotic gate count and circuit latency observed of each design was twice that of a single n-bit quantum accumulator (where the factor of two results from the two out-of-place modular multipliers required for one in-place circuit).

In total, this represents a factor of three reduction in complexity from the typical modular addition approach to quantum modular multiplication, in which each n-bit addition required for product calculation is coupled with two more to perform a modular reduction step. Accordingly, the number of ancilla qubits required for each modular multiplier is principally determined by the requirements of the n-bit addition circuit implemented, with all three of the proposed circuits requiring only a logarithmic number of ancilla beyond those required of for modular-addition-based circuit. Further, in contrast to many previous proposals for fast modular multiplication [4, 5, 14], the proposed algorithms do not rely on inexact mathematical calculations or assumptions, and outperform the modular-adder design for input sizes as low as $n = 2^5$.

Second, the modular multipliers introduced here, and the Barrett and Montgomery circuits in particular, present a unique amenability to implementation with quantum Fourier-basis arithmetic. All three of the proposed modular multipliers can be implemented with quantum Fourier-basis adders with $2n + \mathcal{O}(\log n)$ qubits, matching asymptotically the low-qubit modular-addition circuit proposed in [8]. Assuming equal cost of each quantum gate, the total gate count of all three circuits approaches $2n^2$ at large n, again determined by the single multiplication procedure. Further, the Barrett and Montgomery reduction circuits circumvent the costly Fourier-basis comparisons that dominate circuit latency of the modular addition circuit (and, to a lesser extent, the division-based circuit introduced here). Experimentally, both circuits were demonstrated with latencies just below the analytically-determined $14n$-gate worst-case depth. Comparatively, the Fourier-basis multiplier constructed from modular adders has $\mathcal{O}(n^2)$ latency and requires $\mathcal{O}(n^2 \log n)$ gates, while the faster circuit introduced in [14] requires $9n$ qubits and has a depth of $1000n$, and the inexact multiplier introduced in [5] has the slightly smaller $12n$-gate depth but requires $3n$ total qubits.

Finally, our analysis demonstrates the competitiveness of Fourier-basis arithmetic for realistic (fault-tolerant) quantum modular multiplication. The arbitrary rotations composing Fourier-basis operations can not be implemented directly on CSS codes, but instead must be decomposed into a sequence of available operations. Given a reasonable set of architectural assumptions and the performance bounds for approximating arbitrary quantum gates presented in [32], we nonetheless find that Fourier-basis implementations of the proposed Barrett and Montgomery multipliers can be demonstrated which perform comparably to the equivalent implementations with fault-tolerant logarithmic-depth binary adders. After optimizing specifically for the decomposed Fourier rotation gates, with the assistance of a computerized scheduler, the Fourier-basis multipliers had less than twice the latency of the binary circuits in our model, in exchange for the 60% reduction in required qubits. Gate counts for fault-tolerant Fourier-basis circuits continued to dominate their binary counterparts by a logarithmic factor, but remained within a factor of three in the $8 \leq n \leq 2048$ range of input sizes modeled.

8 Conclusions and Future Work

We have presented three novel techniques for high-performance quantum modular multiplication, adapting fast classical frameworks for division, Montgomery residue arithmetic, and Barrett reduction to efficient reversible procedures. Our techniques are independent of the lower-level implementation of binary quantum addition, and therefore can make use of any current or future quantum addition implementations. As an illustration of this we have constructed and numerically analyzed quantum circuits resulting from three different binary adder implementations. Each modular multiplication technique implements exact, out-of-place modular multiplication with the asymptotic depth and complexity of a single non-modular multiplication, representing a factor of three improvement over the standard quantum modular multiplication technique comprising repeated modular addition. The added gate count and ancilla requirements of our constructions is only $\mathcal{O}(\log n)$. The asymptotic depth and gate count of exact, in-place, controlled modular multiplication (comprising two out-of-place modular multipliers and $3n$ controlled-shift gates) is therefore that of $2n$ quantum adders, comparable to that previously achieved with inexact binary multipliers [4] and improving the $6n$ adders required of the typical modular-addition approach.

A unique advantage of the modular multipliers introduced in this work is their particular amenability to quantum Fourier-basis arithmetic. All three proposed circuits require only $2n + \mathcal{O}(\log n)$ qubits when implemented with Fourier-basis operations, asymptotically matching the low ancilla requirements of Beauregard's Fourier-basis modular-addition-based multiplier [8]. Both the Barrett and Montgomery reduction techniques circumvent the need for repeated comparison operations, and therefore the corresponding QFT and QFT† circuits, which dominate the depth and complexity in the modular addition approach, are not required. Taking advantage of the gate parallelization afforded by Fourier-basis arithmetic, both circuits can then be constructed with an asymptotic depth of $14n$ two-qubit gates. This compares favorably with the $1000n$-gate latency of the fastest prior exact Fourier-basis modular multiplier [14], and is comparable to the $12n$-gate latency of the fastest inexact circuit [5]. Crucially, both prior circuits also expand the ancilla cost of Beauregard's circuit, asymptotically requiring $9n$ and $3n$ qubits, respectively.

Direct comparison between quantum Fourier-basis and binary arithmetic circuits is generally difficult for fault-tolerant systems, as the resource cost of arbitrarily-angled Fourier-basis rotations and Toffoli gates depends highly on the underlying quantum computing hardware and error correction strategy employed. It remains an open question as to whether the efficiency and speedup afforded by Fourier-basis circuits will be applicable to real quantum systems. However, in Section 7 we have shown that with reasonable architectural assumptions, Fourier-basis modular multipliers can be constructed with performance comparable to the fastest binary adder. The space-time tradeoff between the two types of addition circuits is roughly equivalent, with the Fourier-basis adders requiring fewer qubits but more total gates.

In this work, we have primarily discussed circuits for *quantum-classical* modular multiplication, where one of the two input multiplicands is a classical value known at "compile time." Certain important quantum applications, such as the breaking of elliptic-curve cryptographic keys [24], instead require the implementation of in-place, *quantum-quantum* modular multiplication. In the circuits we have presented, only the initial Multiplication and final Uncomputation accumulators depend on the individual input multiplicands, while the Reduction stage differentiating each circuit acts only on the computed product register. An *out-of-place* quantum-quantum modular multiplier is then easily derived by adapting these accumulators to a second quantum input, as described in Section 2. However, in order to construct an in-place modular multiplier from this operator, we now require the multiplicative modular inverse of a quantum input state. Reversible circuits implementing the extended Euclidean algorithm have been demonstrated but overwhelmingly dominate the complexity of the operation [24]. We have shown in the context of modular multiplication that with the adaptation of numerical or representational techniques we can mitigate the overhead of reversible reduction operations. As Euclid's algorithm predominately comprising the sequential calculation of quotients and remainders, the techniques applied here present a similar and potentially significant opportunity for improving the implementation of this operation and corresponding class of problems.

Finally, the modular multipliers we have introduced are not limited to the arithmetic techniques discussed in this paper. For example, the techniques for inexact computation [4, 5], fast large-number multiplication [4], or parallel arithmetic over a superlinear number of qubits [16, 17, 18] could be applied independently to our proposed frameworks. Similarly, the techniques could be extended to different domains; for example, implementations of both Barrett and Montgomery reduction over the Galois field $GF(2^m)$ (critical to elliptic curve cryptography) are well-known classically.

9 Acknowledgments

The authors would particularly like to acknowledge Kevin Obenland at MIT Lincoln Laboratory, whose invaluable discussions, insight, and expertise in both the design of high-performance reversible arithmetic and the efficient computational analysis of reversible circuits was critical to the success of this work. We also graciously acknowledge the support of the NSF iQuISE IGERT and the Intelligence Advanced Research Projects Activity (IARPA).

References

[1] P. L. Montgomery. Modular Multiplication without Trial Division. *Mathematics of Computation*, 44(170):519–521, 1985.

[2] Paul Barrett. Implementing the Rivest, Shamir, and Adleman public key encryption algorithm on a standard digital signal processor. *Advances in Cryptology*, 263:311–323, 1987.

[3] P. W. Shor. Algorithms for Quantum Computation: Discrete Logarithms and Factoring. In *Proceedings 35th Annual Symposium on Foundations of Computer Science*, pages 124–134, 1994.

[4] Christof Zalka. Fast versions of shor's quantum factoring algorithm. *arXiv*, 1998. http://arxiv.org/abs/quant-ph/9806084.

[5] Samuel A. Kutin. Shor's algorithm on a nearest-neighbor machine. *arXiv*, 2006. http://arxiv.org/abs/quant-ph/0609001v1.

[6] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, February 1978.

[7] Thomas G. Draper. Addition on a quantum computer. *arXiv*, 2000. http://arxiv.org/abs/quant-ph/0008033v1.

[8] Stephane Beauregard. Circuit for Shor's algorithm using 2n+3 qubits. *Quantum Information & Computation*, 3(2):14, March 2002.

[9] David Beckman, Amalavoyal N. Chari, Srikrishna Devabhaktuni, and John Preskill. Efficient networks for quantum factoring. *Phys. Rev. A*, 54:1034–1063, Aug 1996.

[10] Vlatko Vedral, Adriano Barenco, and Artur Ekert. Quantum networks for elementary arithmetic operations. *Phys. Rev. A*, 54:147–153, Jul 1996.

[11] 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 & Computation*, 4(4):237–251, July 2004.

[12] Steven A. Cuccaro, Thomas G. Draper, Samuel A. Kutin, and David Petrie Moulton. A new quantum ripple-carry addition circuit. *arXiv*, 2004. http://arxiv.org/abs/quant-ph/0410184.

[13] Thomas G. Draper, Samuel A. Kutin, Eric M. Rains, and Krysta M. Svore. A logarithmic-depth quantum carry-lookahead adder. *Quantum Information & Computation*, 6(4&5):351–369, 2006.

[14] Archimedes Pavlidis and Dimitris Gizopoulos. Fast quantum modular exponentiation architecture for shor's factoring algorithm. *Quantum Information & Computation*, 14(7-8):649–682, 2014.

[15] C.S. Wallace. A suggestion for a fast multiplier. *Electronic Computers, IEEE Transactions on*, EC-13(1):14–17, Feb 1964.

[16] Phil Gossett. Quantum carry-save arithmetic. *arXiv*, 1998. arXiv:quant-ph/9808061.

[17] Rodney Van Meter and Kohei M. Itoh. Fast quantum modular exponentiation. *Phys. Rev. A*, 71:052320, May 2005.

[18] Paul Pham and Krysta M. Svore. A 2d nearest-neighbor quantum architecture for factoring in polylogarithmic depth. *Quantum Information & Computation*, 13(11-12):937–962, 2013.

[19] A. Karatsuba and Y. Ofman. Multiplication of Many-Digital Numbers by Automatic Computers. *Proceedings of the USSR Academy of Sciences*, 145:293–294, 1962.

[20] A. Schönhage and V. Strassen. Schnelle multiplikation großer zahlen. *Computing*, 7(3-4):281–292, 1971.

[21] Martin Fürer. Faster Integer Multiplication. *SIAM Journal on Computing*, 39(3):979–1005, January 2009.

[22] Luis Antonio Brasil Kowada, Renato Portugal, and Celina Miraglia Herrera de Figueiredo. Reversible karatsuba's algorithm. *Journal of Universal Computer Science*, 12(5):499–511, jun 2006.

[23] Adriano Barenco, Artur Ekert, Kalle-Antti Suominen, and Päivi Törmä. Approximate Quantum Fourier Transform and Decoherence. *Physical Review A*, 54(1):14, July 1996.

[24] John Proos and Christof Zalka. Shor's discrete logarithm quantum algorithm for elliptic curves. *Quantum Info. Comput.*, 3(4):317–344, July 2003.

[25] C.H. Bennett. Logical reversibility of computation. *IBM Journal of Research and Development*, 17(6):525–532, Nov 1973.

[26] Xiao-Qi Zhou, Timothy C. Ralph, Pruet Kalasuwan, Mian Zhang, Alberto Peruzzo, Benjamin P. Lanyon, and Jeremy L. O'Brien. Adding control to arbitrary unknown quantum operations. *Nature Communications*, 2, August 2011.

[27] Edward Fredkin and Tommaso Toffoli. Conservative logic. *International Journal of Theoretical Physics*, 21(3):219–253, Apr 1982.

[28] Alfred J. Menezes, Scott A. Vanstone, and Paul C. Van Oorschot. *Handbook of Applied Cryptography.* CRC Press, Inc., Boca Raton, FL, USA, 1st edition, 1996.

[29] R. Cleve and J. Watrous. Fast parallel circuits for the quantum Fourier transform. In *Proceedings 41st Annual Symposium on Foundations of Computer Science*, pages 526–536. IEEE Comput. Soc, 2000.

[30] Cristopher Moore and Martin Nilsson. Parallel Quantum Computation and Quantum Codes. *SIAM Journal on Computing*, 31(3):799–815, January 1998.

[31] A. M. Steane. Error correcting codes in quantum theory. *Phys. Rev. Lett.*, 77:793–797, Jul 1996.

[32] N. J. Ross and P. Selinger. Optimal ancilla-free Clifford+T approximation of z-rotations. *ArXiv e-prints*, March 2014.

[33] Ivan Kassal, Stephen P Jordan, Peter J Love, Masoud Mohseni, and Alán Aspuru-Guzik. Polynomial-time quantum algorithm for the simulation of chemical dynamics. *Proceedings of the National Academy of Sciences of the United States of America*, 105(48):18681–18686, August 2008.

[34] Milos D. Ercegovac and Tomas Lang. *Digital Arithmetic.* Amsterdam; Boston, MA: Morgan Kaufmann/Elsevier., 2004.

[35] M. A. Nielson and I L Chuang. *Quantum Computation and Quantum Information.* Cambridge University Press, 2002.

[36] Julio T Barreiro, Markus Müller, Philipp Schindler, Daniel Nigg, Thomas Monz, Michael Chwalla, Markus Hennrich, Christian F Roos, Peter Zoller, and Rainer Blatt. An open-system quantum simulator with trapped ions. *Nature*, 470(7335):486–491, February 2011.

A Implementation of quantum arithmetic

Integer quantum adders and multipliers are the base underlying circuits for all the circuit constructions described in this paper. Because an integer multiplier can be constructed from repeated controlled integer adders, the integer addition circuit can be considered the fundamental circuit of all the constructions. The basic modular multiplier constructed from modular adders requires only one type of adder circuit, while the Barrett and Montgomery modular multipliers require additional adders in the reduction circuitry. These adders have different widths compared to the adders used in the main multiplier, and therefore, it may be advantageous to utilize several different types of adders in these circuits.

Because the adder is such a fundamental circuit in all our circuit constructions the design of each adder used will have a significant impact on the design and resources required by our modular multiplier circuits. Many quantum adders have been proposed, and we summarize the ones used in our circuit constructions in Table 7. The adders fall into two main categories: adders based on reversible implementations of classical adders, and adders that operate in a transformed Fourier basis. Each of these adders presents a different trade-off with respect to the total gates required, the circuit depth, and the number of ancilla used. These resource trade-offs translate directly to the multiplier designs, however, the form of the adders can also impact the resource requirements of the multipliers. For example, the Fourier adders allow gates from multiple adders to be overlapped, which can reduce the overall depth of the multiplier.

Table 7: Quantum adder circuits used in multiplier constructions. The resource requirements are assuming the in-place addition of a classical value onto a quantum register of width n, and are given to leading order only. The resources for the Fourier transform basis adder assume the decomposition of the rotation gates required to a specified accuracy (ϵ) using a technique such as described in [32].

Adder type	**Toffoli/T depth**	**Toffoli/T gates**	**Qubits required**
Majority ripple [12]	$2n$	$2n$	$2n+1$
Prefix-ripple [Section A.1]	n	$3n$	$2n+1$
Carry look-ahead [13]	$4log_2(n)$	$10n$	$4n - log_2(n) - 1$
Fourier transform basis [33]	$3log_2(1/\epsilon)$	$3nlog_2(1/\epsilon)$	n

A.1 Prefix Adders

The calculation of the carry bit-string for an adder can be thought of as a prefix operation, i.e., $C = c_{n-1} \circ \cdots \circ c_2 \circ c_1 \circ c_0$. Where each c_i is represented by the tuple (p_i, g_i) and p_i/g_i indicates that a carry is propagated/generated at position i. The prefix composition function is defined as: $(p_{ij}, g_{ij}) = (p_i \wedge p_j, g_j \vee (g_i \wedge p_j))$. For a multi-bit adder with inputs $a_{[n-1:0]}$ and $b_{[n-1:0]}$ the single-bit inputs to the prefix network are calculated as: $p_i = a_i \oplus b_i$ and $g_i = a_i \wedge b_i$. The generate value from the first bit to bit i (defined as g_{0i}) is the carry out of position i. For a multi-bit adder, a parallel network can be used to compute the prefix bits. For classical, non-reversible, adders many networks have been proposed and used to create adders. Two example parallel-prefix networks are shown in Figure 24. For a description of these adders and others consult any textbook on computer arithmetic, for example [34].

Reversible adders that are suitable for quantum computing can be constructed from parallel prefix networks, however, because of the constraints of reversible logic, which adders require the fewest resources and have the lowest depth may be different than in the classical non-reversible case. For example the adder described in [13] is based on the network structure shown in Figure 24(a). The depth of this adder is logarithmic in the number of bits and is well suited for a reversible implementation because it uses low fan-in/out nodes and requires fewer gates than many of the other proposed log-depth networks.

Linear-depth parallel-prefix networks can also be defined, for example the network shown in Figure 24(b) has depth $n/2+1$ for n bits. This adder is similar to a sequential ripple adder, but because we calculate two-bit propagate values across adjacent pairs of bits the carries can be rippled across two bits at each step. The odd-position carries are not in the critical path of the circuit and can be computed in a later step. We call this adder the prefix-ripple adder. We have implemented a reversible circuit based

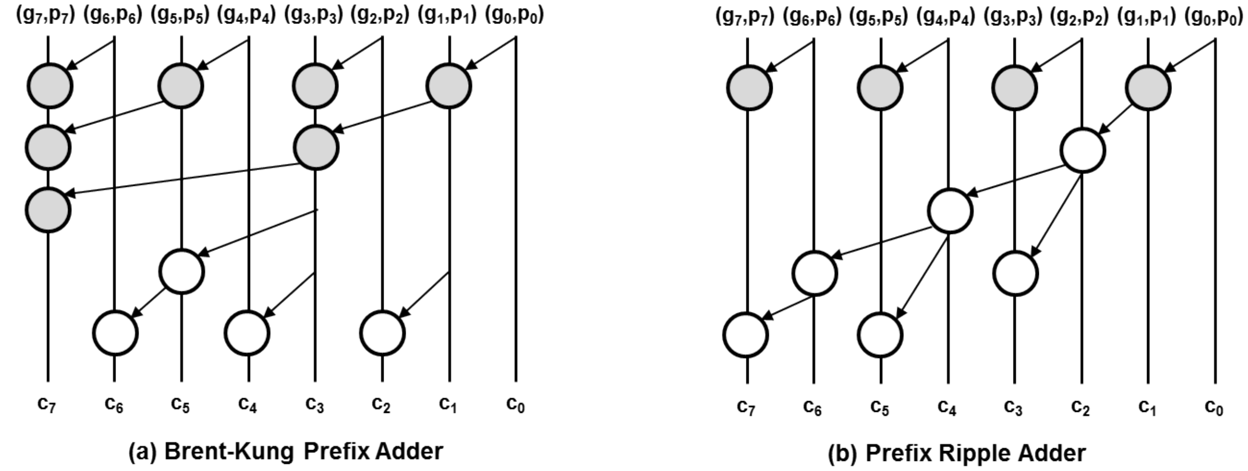

(g7,p7) (g6,p6) (g5,p5) (g4,p4) (g3,p3) (g2,p2) (g1,p1) (g0,p0)
c7 c6 c5 c4 c3 c2 c1 c0
(a) Brent-Kung Prefix Adder
(g7,p7) (g6,p6) (g5,p5) (g4,p4) (g3,p3) (g2,p2) (g1,p1) (g0,p0)
c7 c6 c5 c4 c3 c2 c1 c0
(b) Prefix Ripple Adder

Figure 24: Network structure for two different prefix adders. The shaded nodes produce both propagate (p) and generate (g) bits from the inputs and the un-shaded nodes only produce the generate bits. The Brent-Kung adder has depth $2log_2(n) - 1$ for n bits and the prefix-ripple adder has depth $n/2 + 1$.

on the network of Figure 24(b), and the repeating segment used to calculate pairs of carries is shown in Figure 25. The first section of the circuit calculates the 2-bit propagate values and can be executed in parallel across all bits in the adder. $n/2$ sequential TOFFOLI gates are then used to ripple the carry to all even bit-positions of the adder. The last step is to calculate the odd-position carries. The carry at position i can be calculated in parallel with the calculation of an even-position carry in a later step, and therefore only adds a TOFFOLI depth of one to the entire circuit. A full circuit built from the two-bit segments of Figure 25 would require 4 TOFFOLI gates for every two bits in the full out-of-place addition of two quantum values. The circuit also requires an additional n-bit ancilla register to hold the carries. Comparing the prefix-ripple adder to the ripple adder in [12], the new adder has half the TOFFOLI depth but requires a factor of 2 more TOFFOLI gates, and an extra n-bit ancilla register. If the prefix-ripple adder is used to add a classical value to a quantum one, then the first TOFFOLI gate in the two-bit segment is reduced in degree and the cost becomes 1.5 TOFFOLI gates per bit. Additionally the total number of qubits for this classical/quantum adder is $2n + 1$, which is equivalent to that required by the adder in [12] used in the same way.

A.2 Select undo adder

The adders descried in this section thus far have assumed unconditional addition of two values. However, normally we require an adder that includes a global control. For example, this is the case when the adder is used in a multiplier circuit. Typically a reversible in-place addition first calculates $|x\rangle|y\rangle|0\rangle \longrightarrow |x\rangle|y\rangle|x+y\rangle$ out-of-place and then because the subtraction: $|x\rangle|x+y\rangle|0\rangle \longrightarrow |x\rangle|x+y\rangle|y\rangle$ produces the same set of outputs, we can run it in reverse after the addition to produce our in-place addition: $|x\rangle|y\rangle|0\rangle \longrightarrow |x\rangle|x+y\rangle|0\rangle$. Because $|y\rangle$ and $|x+y\rangle$ are at positions 2 and 3 respectively after the addition, but the subtraction requires that they are in the opposite order, we must swap the two registers. However, in most cases we can just relabel the two registers instead of swapping them.

The similarity between addition and subtraction suggests a way to perform controlled in-place addition. A controlled adder should perform the in-place addition as described above when the control bit is set, but when the bit is reset, it should undo the action of the addition. In the first case we are performing a reverse subtraction, and in the later case a reverse addition. We can easily construct a circuit that selects between these two cases based on the state of a control bit. A subtraction is just an addition where one of the inputs has been negated. In two-complemented arithmetic this can be done by selectively flipping each bit with a CNOT gate and setting the input carry to one. The SWAP between

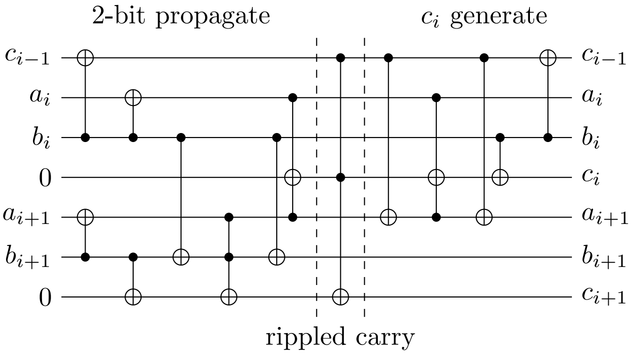


Figure 25: Circuit to calculate two bits of the carry for the prefix-ripple adder. This adder requires 4 TOFFOLI gates when both inputs are quantum values. The section labeled *2-bit propagate* can be executed in constant time across all the bits in the adder. The *rippled carry* step requires $n/2$ steps for the entire adder and calculates the second carry bit in the pair. The first carry bit of the pair is calculated after rippling the carry and can be done in constant time for all bits in the adder.

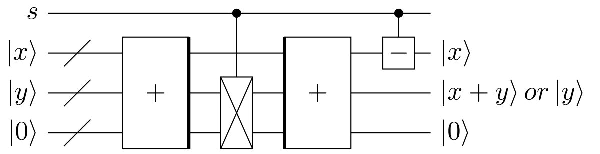


Figure 26: Inplace select undo adder. When the select bit is 0 the second reverse adder uncomputes the first addition. When the select bit is 1 the second addition acts as a reverse subtraction clearing the $|y\rangle$ input.

the two out-of-place adders must now be controlled, requiring two CNOT gates and one TOFFOLI per bit. This controlled adder, which we call the *select undo* adder, is illustrated in Figure 26. For this adder neither of the out-of-place adders is controlled and, depending on the type of adder employed, this may lead to a reduction in the number of TOFFOLI gates required. The main extra cost of the select-undo adder are the n TOFFOLI gates required between the two out-of-place adders. However, the SWAP that they are used to implement is between two existing registers and therefore no extra ancillas are required.

A.3 Quantum Multiplication

Given an in-place quantum adder, we can a construct quantum multiply-accumulate circuit,

$$|z\rangle_w |y\rangle_n \xrightarrow{\text{Q-MAC}(X)} |Xy + z\rangle_w |y\rangle_n . \tag{61}$$

Using the input $|z\rangle$ as an initialized accumulator register, we add $2^k X$ in-place for each bit y_k of $|y\rangle_n$, requiring n controlled, in-place quantum adders. The Q-MAC operator can be generalized to any sum controlled by the bits of $|y\rangle$. In particular, we can accumulate the congruent value,

$$t \triangleq \sum_{k=0}^{n-1} y_k \big(2^k X \bmod N\big), \tag{62}$$

such that $t \equiv Xy \pmod N$ and requires at most $\lceil \log_2 nN \rceil \leq n + \lceil \log n \rceil$ bits to represent. As before, we require n in-place additions controlled by the bits of $|y\rangle_n$ to an accumulation register $|z\rangle_w$, now of the reduced partial products $2^k X \bmod N$. If the accumulator is smaller than is required to hold t (i.e. $w < \lceil \log_2(nN) \rceil$), these adders are truncated and the resulting product is computed modulo-2^w.

For classical X, we can also construct an in-place quantum multiplier,

$$|0\rangle_w |y\rangle_n \xrightarrow{\text{Q-MUL}(X)} |Xy\rangle_{n+w} , \tag{63}$$

where the product is computed over the input register $|y\rangle$ and is implicitly modulo-2^{n+w}. For odd X, we can express the product,

$$Xy = y + y(X-1) = y + \sum_{k=0}^{n-1} y_k \cdot 2^{k+1} \left(\frac{X-1}{2} \right), \tag{64}$$

where $(X-1)/2$ is an integer. Each addition of $2^{k+1}(X-1)/2$ is then conditioned on the kth bit of y, and affects only the bits more significant than k in the resulting sum. Given a register initialized to $|y\rangle_n$, we can therefore perform each of these additions in-place in descending order ($k = n-1, ..., 0$), so that each bit y_k of $|y\rangle$ controls an addition affecting only the more significant bits of the register before it is affected by any addition in the sequence.

For even X, the addend $(X-1)/2$ is not an integer. Instead, we compute the equivalent product $(X/2^\lambda)y$, where $\lambda = v_2(X)$ is the two-adic order of X. The full product Xy can then be produced by simply concatenating this result with λ zero-initialized ancilla bits. The in-place multiplier relies on the k trailing zeros of each partial product $2^k X$, and so is not compatible with the partially-reduced multiply introduced above. However, given the distribution of trailing zeros in the set of reduced partial products, it is likely that the result can be computed over about $\log_2(n)$ bits of the input state.

A.4 Addition in Fourier transform basis

Central to quantum Fourier-basis arithmetic is the Fourier number state representation. Defining the single-qubit state,

$$\begin{aligned} |\Phi(\alpha)\rangle &\triangleq \cos(\alpha\pi/2)\,|0\rangle + \sin(\alpha\pi/2)\,|1\rangle \\ &= \mathrm{Y}(\alpha\pi)\,|0\rangle , \end{aligned} \tag{65}$$

where $\mathrm{Y}(\alpha) = e^{-i\alpha\sigma_y/2}$ represents the single-qubit Y-axis rotation, the Fourier representation of an n-bit number x is defined by the product state,

$$|x\rangle_n^\Phi \triangleq \bigotimes_{k=0}^{n-1} \left|\Phi\left(\frac{x}{2^k}\right)\right\rangle. \tag{66}$$

Note that this differs from the typical [35] definition, as defined as $|x\rangle_n{}^{\Phi'} \triangleq \sum_{j=0}^{2^n-1} e^{-ijx\pi/2^n} |j\rangle_n$. The latter can be recovered as $\mathrm{H}^{\otimes n} S^{\otimes n} |x\rangle_n^\Phi$, where $\mathrm{H}^{\otimes n}$ and $S^{\otimes n}$ indicate global Hadamard and phase gates, respectively.

Uncontrolled, in-place addition of a classical parameter to a n-bit quantum Fourier register $|x\rangle_n^\Phi$ requires n unconditional Y-rotations,

$$\begin{aligned} |x+y\rangle_n^\Phi &= \bigotimes_{k=0}^{n-1} \left|\Phi\left(\frac{x+y}{2^k}\right)\right\rangle \\ &= \bigotimes_{k=0}^{n-1} \mathrm{Y}\left(\frac{y\pi}{2^k}\right) \left|\Phi\left(\frac{x}{2^k}\right)\right\rangle \\ &= \left[\prod_{k=0}^{n-1} \mathrm{Y}^{(k)}\left(\frac{y\pi}{2^k}\right)\right] |x\rangle_n^\Phi, \end{aligned} \tag{67}$$

where $\mathrm{Y}^{(k)}(\alpha)|x\rangle_n$ indicates a Y-rotation of the kth qubit of $|x\rangle_n$. Controlled Fourier addition of a classical parameter then requires conditioning each of these rotation gates on a single control qubit, inhibiting parallelization of the standalone adder.

Instead, the commutability of the Fourier rotations (and lack of carries to propagate) admits large-scale parallelization for multiple additions. In particular, we can construct an out-of-place multiply-accumulate operator as usual, in which each bit of a binary input register $|x\rangle_n$ controls the Fourier addition of a classical value to an accumulation register $|P\rangle_w^\Phi$ in Fourier representation. The rotations required by this sum can be rearranged and executed in parallel, with a total depth of at most $\max(w, n)$ gates.

In order to construct a quantum-quantum Fourier adder,

$$|x\rangle_n^\Phi |y\rangle_n \longrightarrow |x+y\rangle_n^\Phi |y\rangle_n, \tag{68}$$

we must construct the $\mathrm{Y}^{(k)}(y\pi/2^k)$ rotations bitwise, performing the set of conditional rotations $\mathrm{Y}^{(k)}(2^l y_l/2^k)$ for each bit y_l of $|y\rangle_n$. That is, the quantum-quantum Fourier adder is simply a special case of the Fourier multiply-accumulate operation, for which the multiplier is one.

Finally, observing Equation (67), any addition of classical values $y \ll 2^n$ will involve asymptotically small rotations on bits more significant than $k \sim \lceil \log_2 y \rceil$. As in [36], these operations can therefore be truncated $\mathcal{O}(\log ny)$ gates with negligible loss of, or possibly improved, fidelity.

A.5 Quantum Fourier Multiplication

Somewhat counter-intuitively, given a binary input state $|y\rangle_n$, we can also perform an in-place multiply using quantum Fourier adders. Observing the kth bit of the Fourier state $|Xy\rangle^\Phi$ (again assuming odd X),

$$\left|\Phi\left(\frac{Xy}{2^k}\right)\right\rangle = \left|\Phi\left(y_k + \sum_{j<k} \frac{y_j 2^j X}{2^k}\right)\right\rangle = \pm \prod_{j<k} \mathrm{Y}\left(\frac{y_j 2^j X}{2^k}\right) |y_k\rangle, \tag{69}$$

we find the binary input bit $|y_k\rangle$, rotated by the less significant qubits in the register. Again beginning with the MSB, we perform in-place additions of $X/2$ controlled by each bit y_k of the input $|y\rangle$ and acting on the more significant bits of the register. The resulting state is the Fourier representation of the product:

$$|y\rangle_n \xrightarrow{\Phi\text{-MUL}(X)} |Xy\rangle_n^\Phi. \tag{70}$$

The quantum Fourier transform, $|y\rangle \longrightarrow |y\rangle^{\Phi}$, is then the special case Φ-MUL(1). Crucially, the Φ-MUL(X) can be parallelized identically to the standalone QFT, with a depth of $2n + \mathcal{O}(1)$ on a variety of computational topologies [29, 30, 18].